File No. 1130-30
Form C26-5929-2

IBM Systems Reference Library

IBM 1130 Subroutine Library

This publication describes the subroutines in the IBM 1130 Subroutine Library. The library consists of input/output, conversion, arithmetic and functional, and selective dump subroutines. Included in the descriptions are calling sequences for the subroutines and explanations of the parameters involved.

The section on conversion subroutines describes the codes used to communicate with the 1130 System input/output devices. An appendix lists the codes, and shows their relationship to each other.

PREFACE

This publication describes how the programmer can use the IBM 1130 Subroutine Library to increase the efficiency of his programs and decrease the writing and testing time. The subroutine library includes the following subroutines.

- Interrupt Service
- Interrupt Level
- Data Conversion
- Arithmetic and Functional
- Selective Dump
- FORTRAN

These subroutines are available for use with both the 1130 Assembler and the 1130 FORTRAN Compiler.

With the assembler, the user calls the subroutines via a calling sequence. The appropriate subroutine calls are generated by the FORTRAN compiler whenever a read, write, arithmetic, or CALL statement is encountered. This publication describes each subroutine and the required calling sequence. All subroutines in the 1130 Subroutine Library are included in the list which appears in Appendix A.

It is assumed that the reader is familiar with the methods of data handling and the functions of instructions in the IBM 1130 Computing System. He must also be familiar with the assembler or compiler used in conjunction with the subroutines. The following IBM publications provide the prerequisite information.

IBM 1130 Functional Characteristics (Form A26-5881)

IBM 1130 Computing System Input/Output Units (Form A26-5890)

IBM 1130 Assembler Language (Form C26-5927)

IBM 1130 FORTRAN Language (Form C26-5933)

MACHINE CONFIGURATION

The use of the subroutine library requires the following machine configuration:

IBM 1131 Central Processing Unit with a minimum of 4096 words of core storage

IBM 1442 Card Read Punch, or IBM 1134 Paper Tape Reader with IBM 1055 Paper Tape Punch

In addition, the following input/output units and features can be controlled by the input/output section of the subroutine library:

Console Printer/Input Keyboard
Disk Storage
IBM 1132 Printer
IBM 1627 Plotter

This edition incorporates the information from, but does not obsolete, the previous edition (C26-5929-1) as amended by the following Technical Newsletters:

Form No.	Pages	Date
N26-0551	iii-iv,41-42,45-46,46.1-46.2,46.3	1/18/66
N26-0553	iii-iv,1-2,5-6,9-10,11-12,13-14,19-20, 21-22,23-24,27-28,33-34,46.3-46.4, 47-48,49-50	3/1/66

Copies of this and other IBM publications can be obtained through IBM Branch Offices. A form has been provided at the back of this publication for readers' comments. If the form has been detached, comments may be directed to: IBM, Programming Publications Dept. 452, San Jose, Calif. 95114

CONTENTS

INTRODUCTION

It is often necessary to repeat a group, or block, of instructions many times during the execution of a program (examples include conversion of decimal values to equivalent binary values, computation of square roots, and reading data from a card reader). It is not necessary to write the instructions each time a function is required. Instead, the block of instructions is written once, and the main program transfers to that block each time it is required. Such a block of instructions is called a subroutine. Subroutines normally perform such basic functions that they can assist in the solution of many different kinds of problems.

When a main program uses a subroutine several times, which is the common situation, the block of instructions constituting the subroutine need appear only once. Control is transferred from a main program to the subroutine by a set of instructions known as a calling sequence, or basic linkage. A calling sequence transfers control to a subroutine and, through parameters, gives the subroutine any control information required.

The parameters of a calling sequence vary with the type of subroutine called. An input/output subroutine requires several parameters to identify an input/output device, storage area, amount of data to be transferred, etc.; whereas an arithmetic/functional subroutine usually requires one parameter representing an argument. Each calling sequence used with the 1130 System subroutines consists of a CALL or LIBF statement (whichever is required to call the specific subroutine), followed by DC statements that make up the parameter list. The calling sequences for the various subroutines in the subroutine library are presented later in the manual. Each subroutine is self-contained, so that only those routines required by the current job are in core storage at program execution time.

The interrupt service subroutines (ISS) transfer data from and to the various input/output devices attached to the computer. The subroutines handle all of the details peculiar to each device, including the usually complex interrupt functions, and can control input/output devices simultaneously and asynchronously.

ISS CHARACTERISTICS

To fully comprehend subsequent descriptions of each ISS, the user should be familiar with the following characteristics, which are common to all ISS:

- Methods of data transfer
- Interrupt processing
- ILS (interrupt level subroutine)
- ISS operation
- General error handling procedures
- Basic calling sequence

METHODS OF DATA TRANSFER

IBM 1130 I/O devices and their related subroutines can be differentiated according to their methods of transmitting and/or receiving data.

Direct Program Control

The serial I/O devices operate via direct program control, which requires a programmed I/O operation for each word or character transferred. A character interrupt occurs whenever a character I/O operation is completed. Direct program control of data transfer is used for the serial devices including the card read punch, paper tape reader and punch, console printer, input keyboard, 1132 Printer, and plotter.

Data Channel

Disk storage operates via a data channel, which requires an I/O operation only to initiate data transfer. A device is provided with control information, word-counts, and data from the user's I/O area. Once initiated, data transfer proceeds asynchronously to program execution. An operation-complete interrupt signals the end of an I/O operation when all data has been transferred.

INTERRUPT PROCESSING

Interrupt processing is divided into two parts, level processing and device processing. The flow of logic in response to an interrupt is: user program interrupted, level processing begun, device processing begun and completed, level processing completed, and user program continued.

Level Processing

Level processing consists of selecting the correct device processing routine, performing certain housekeeping functions, and clearing the level by a BOSC instruction when interrupt processing is complete.

Level processing is done by the ILS (interrupt level subroutines). Entered by interrupts, ILS give temporary control to a device processing subroutine (ISS) and eventually return control to the user program. The interrupt entrance address is stored, at load time, in the appropriate interrupt branch address; location 8 for interrupt level zero (ILS 00), location 9 for interrupt level one (ILS 01), . . . , location 12 for interrupt level four (ILS 04). The device processing entrance address is computed at load time from identifying information, stored in the ILS, in the compressed ISS header card, and in the loader interrupt transfer vector.

Device Processing

Device processing consists of operating an I/O device, processing the interrupts, and clearing the device by an XIO (sense DSW) instruction when interrupt processing is complete.

Device processing is done by the ISS (interrupt service subroutines). They can be entered by a calling instruction (LIBF or CALL), which either requests certain initialization to be done or requests an I/O device operation. They can also be entered by the ILS as part of the interrupt processing. The calling entry point is specified by an ISS statement.

The interrupt entry point(s) is set up in the ISS and identified in the ILS. It is entered indirectly through a branch address table within the ILS.

INTERRUPT LEVEL SUBROUTINES

The ISS package services all input/output interrupts with a set of ILS (interrupt level subroutines), loaded as part of the subroutine library.

Description

There is one ILS for each interrupt level used. Each routine determines which device on its level caused a particular interrupt; preserves the contents of the accumulator, the accumulator extension, index register one (XR1), and the Carry and Overflow indicators; and transmits identifying information to the ISS.

Interrupt service subroutines are loaded first so that the loader loads only the ILS that are required. For example, if a main program does not call the 1132 printer subroutine, the routine for interrupt level 1 need not be loaded because no interrupts will occur on that level.

When the ILS are loaded, the core addresses assigned to them are inserted into the computer words, reserved for that purpose, starting at word 8. Interrupts occurring during execution of a user program cause an automatic Branch Indirect, via the interrupt level word, to the correct ILS.

Recurrent Subroutine Entries

Recurrent entries to a subroutine can result from subsequent interrupts. For example, during execution of the console printer subroutine, a disk interrupt can start execution of a subroutine to handle the condition that caused the disk interrupt. If this handling includes calling the console printer subroutine, certain information is destroyed, the most important of which is the return address of the program that originally called the console printer.

To prevent the loss of data resulting from a recurrent entry, the user must provide the programming required to save the return address and any other data needed to continue an interrupted subroutine after an interrupt has been serviced. The information needed for such programming must be obtained from the subroutine listings and flowcharts.

NOTE: All ISS were written with the assumption that all LIBF's would be executed from the mainline level of interrupt priority. There are <u>no</u> provisions in any ISS to handle recurrent entries.

ISS OPERATION

This section briefly describes the operation of the ISS (interrupt service subroutines). This description, along with some basic flowcharts, should make it easier for the reader to understand the descriptions of individual subroutines presented later.

ISS Subdivision

Each ISS is divided into a call routine and an interrupt response routine. The call routine is entered when a user's calling sequence is executed; the interrupt response routine is entered as a result of an I/O interrupt.

Call Routine

Each ISS saves and restores the contents of the accumulator and extension, index registers, and the Carry and Overflow indicators. The call routine, illustrated in Figure 1, has four basic functions:

1. Determine if any previous operations on the specified device are still in progress.
2. Check the calling sequence for legality.
3. Save the calling sequence.
4. Initiate the requested I/O operation.

The flow diagram (Figure 1) is not exact for any one ISS. It is only a general picture of the internal operation of a call routine.

<u>Determine Status of Previous Operation.</u> This function can be performed by using a programmed routine-busy indicator to determine if a previous operation is complete. The CARD1 subroutine is a good example. When an operation is started on the 1442, a subsequent LIBF CARD1 for the 1442 is not honored until the routine-busy indicator is turned off. A call to any other ISS subroutine, such as TYPE0, is not affected by the fact that the CARD1 subroutine is busy.

Each ISS, except PAPTN, can use one programmed routine-busy indicator to determine if a previous operation is complete. The PAPTN subroutine uses two busy indicators, one for the paper tape reader and one for the punch. If an operation is started on the reader, a subsequent LIBF PAPTN for the reader is not honored until the Reader Busy indicator is turned off. However, an LIBF PAPTN for the paper tape punch is treated in the same manner as a call to any other ISS and is not affected by the fact that the reader is busy.

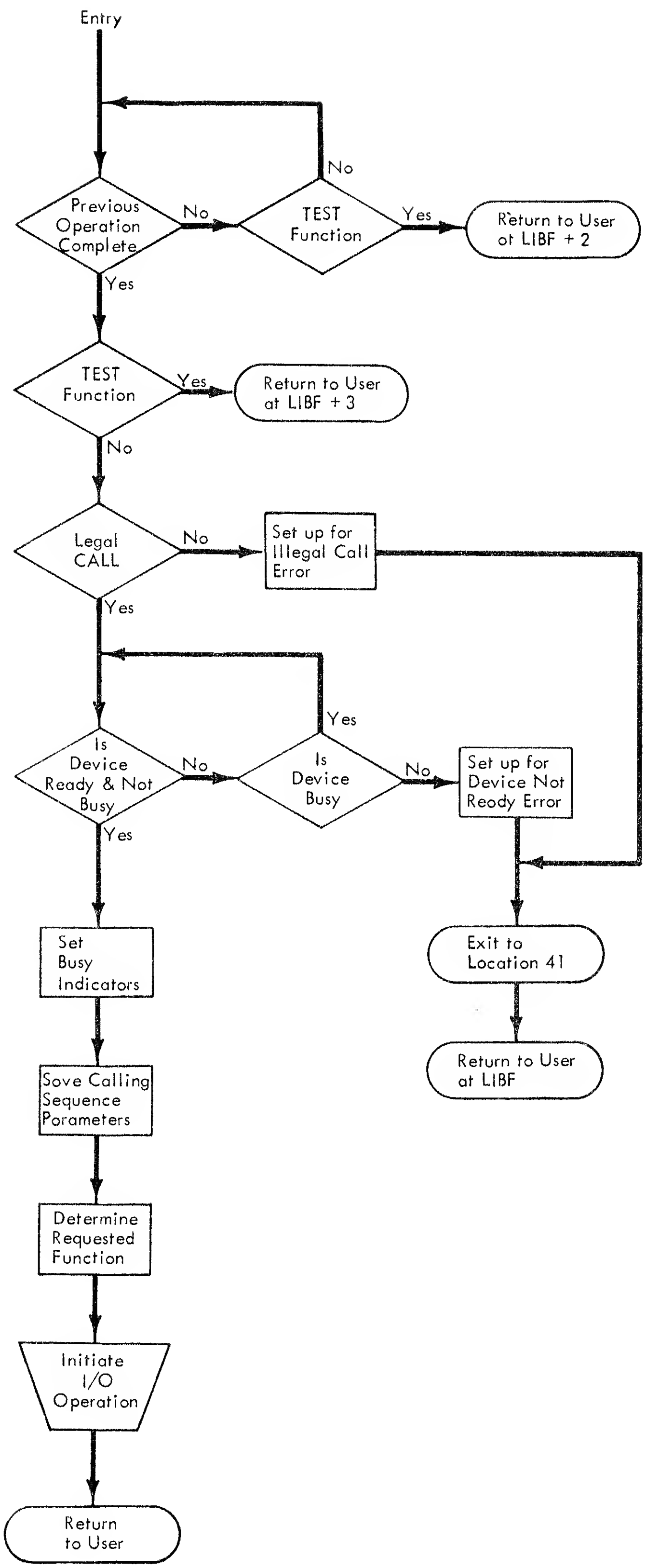


Figure 1. Call Routine

Check Legality of Calling Sequence. Calling sequences are checked for such items as illegal function character, illegal device identification code, zero or negative word count, etc.

Save Calling Sequence. The call routine saves, within itself, all of the calling sequence information needed to perform an I/O operation. The user can modify a calling sequence, even though an I/O operation is not yet complete.

NOTE: The I/O data area should be left intact during an operation because the ISS is continually accessing and modifying that area.

Initiate I/O Operation. The call routine only initiates an I/O operation. Subsequent character interrupts or operation complete interrupts are handled by the interrupt response routine.

Interrupt Response Routine

The I/O interrupt response routine is illustrated in Figure 2.

Operation. An I/O interrupt causes a user program to exit to an interrupt level routine, which in turn exits to the I/O interrupt response routine. The interrupt response routine checks for errors, does any necessary data manipulation, initiates character operations, and initiates retry operations in case of errors. It then returns control to the interrupt level routine, which returns control to the user.

Character Interrupts. These interrupts occur for devices under direct program control whenever data can be read or written, e.g., a card column punched or a paper tape character read.

Operation Complete Interrupts. These interrupts occur in disk and card operations when a specified block of data has been read or written, e.g., a disk record read.

Error Detection and Recovery Procedures. Are an important part of an ISS. However, little can be done about reinitiating an operation until a character interrupt or operation complete interrupt occurs. Therefore, error indicators are not examined until one of these interrupts occurs.

Recoverable Device. This is an I/O device that can be easily repositioned by a subroutine or by an operator and an I/O operation reinitiated. If a device is not recoverable, or if an error cannot be corrected

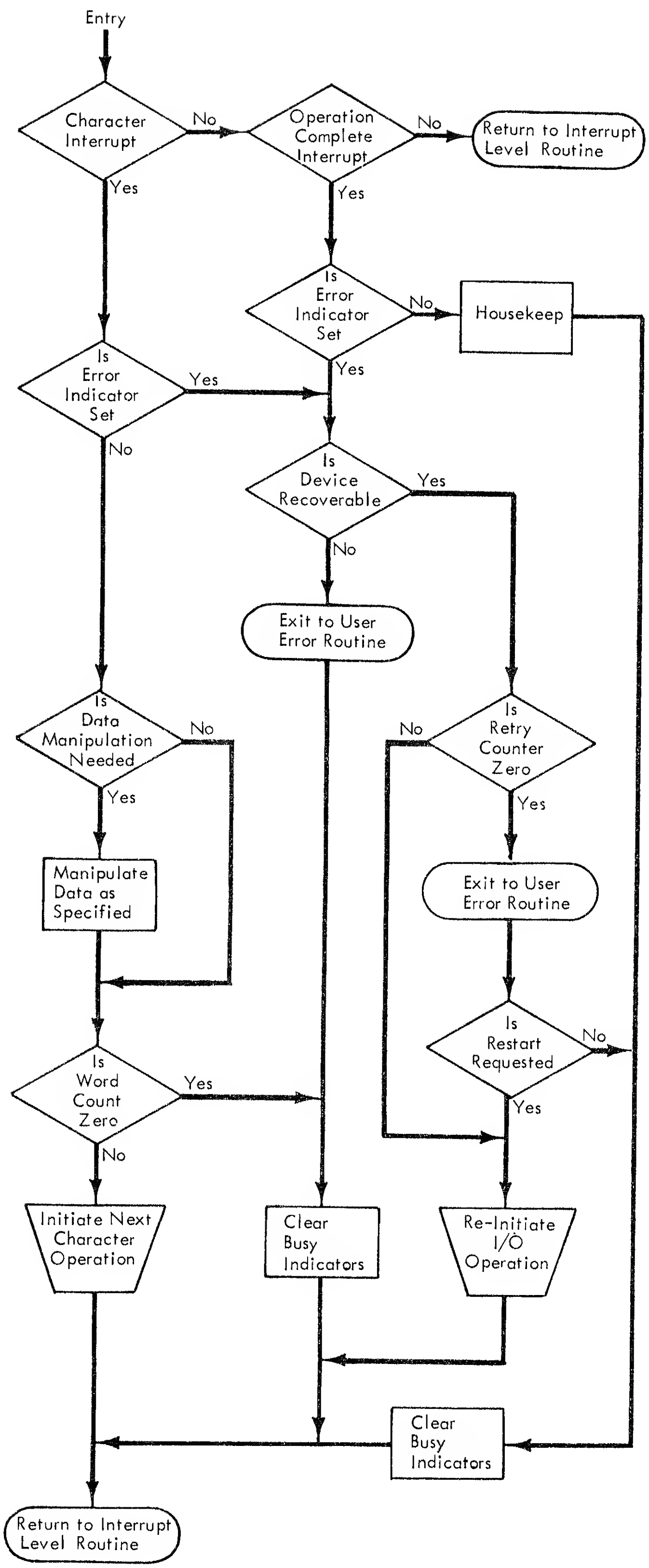


Figure 2. Interrupt Response Routine

after a specified number of retries, the user is informed of the error condition. If a device is recoverable, the user may request, via his error routine, that an operation be reinitiated.

GENERAL ERROR-HANDLING PROCEDURES

Each ISS has its own error detecting routines which categorize the error and choose an error procedure. (In this context, the term, error, includes such conditions as last card, channel 9, channel 12, etc.) Errors fall into one of two categories: those that are detected before an I/O operation is initiated, and those that are detected after an I/O operation has been initiated. Appendix B contains a list of the errors detected by the ISS; Appendix C contains descriptions of the actions taken by each ISS after the return from user-written error subroutines.

Pre-operation Error Detection

Before an ISS initiates an I/O operation, it checks the device status and the legality of calling sequence parameters. If a device is not ready, or a parameter is in error, the subroutine stores the address of the LIBF statement in core location 40 and exits to core location 41. The accumulator is loaded with an error code, represented by four hexadecimal digits, which defines the error (see Appendix B).

Digit 1 identifies the ISS subroutine called:

1 - CARD0 or CARD1

2 - TYPE0 or WRTY0

3 - PAPT1 or PAPTN

5 - DISK0, DISK1, or DISKN

6 - PRNT1

7 - PLOT1

Digits 2 and 3 are not used.

Digit 4 identifies the error

0 - device not ready

1 - illegal LIBF parameter or illegal specification in the I/O area.

The loader stores a Wait instruction in core location 41 and an Indirect Branch instruction (BSC I 40) in locations 42 and 43. Therefore, the LIBF may be executed again (after the error condition has been

corrected) by pressing PROGRAM START on the console. The user can, if he chooses, replace these two instructions with an exit to his own error routine.

Post-Operation Error Detection

After an I/O operation has been started, certain conditions may be detected about which the user should be informed. The conditions might be card jams for which manual intervention is needed before the operation can continue; read checks that have not been corrected after a specified number of retries; or indications of equipment readiness, such as last card or channel 12 indicators. All of these conditions are detected during execution of the I/O interrupt response routine. (See ISS Operation.)

No Error Parameter. If no error parameter is included in the calling sequence that initiated the I/O operation and one of the conditions is detected, the subroutine initiates a Wait procedure (programmed loop), which continues until an operator corrects the detected condition.

Error Parameter Included. If an error parameter is included in the calling sequence, a Branch and Store Instruction Counter instruction (BSI) to the user's error routine specified in the calling sequence is executed. Identifying information is placed in the accumulator and extension (see Appendix B). When the user's error routine returns control to the ISS using the return link (see Basic Calling Sequence), the subroutine examines the accumulator. If the user clears the accumulator before returning to the subroutine, he is requesting that the error condition be ignored and the operation terminated. If the user does not clear the accumulator, he is requesting that the operation be restarted, in which case the subroutine reinitiates the operation before returning to the user's main program.

User's Error Routine Implications. It is important to note that a user's error subroutine (entered via the LIBF error parameter address) is executed as part of the interrupt processing. The interrupt level is still on, preventing recognition of other interrupts of the same or lower priority. This has the following implications:

1. Return must be made to the ISS subroutine via the return link (set up by the BSI instruction executed by the ISS subroutine). Otherwise, normal processing cannot be continued because the ISS subroutine must return to the ILS subroutine to restore the contents of the accumulator and extension, status indicators, and index registers.
2. Return must be made with a BSC instruction, not a BOSC instruction. Otherwise, the interrupt level is turned off, setting up the possibility of another interrupt on the same level destroying the return address to the user from the ILS.
3. An LIBF or CALL to another subroutine from the user's error subroutine can cause a recurrent-entry problem. If that subroutine is already in use when the interrupt occurs, the user's new LIBF or CALL destroys the original return address and disrupts operation of the called subroutine.
4. An LIBF or CALL to another ISS can cause an endless loop if the new I/O device operates on the same or lower priority interrupt level than the device that caused the error.

NOTE: A call to WRTY0 to type an error message can be made only if the user does not then wait for the completion of typing or operator intervention before returning to the ISS.

5. The user should have a separate error subroutine for each device to prevent errors on several deviees (on different levels) from causing recurrent-entry problems in the user's error subroutine.

NOTE: The error codes in the Accumulator do not differentiate between ISS as the preoperative error codes do.

Since the ILS saves XR1 as part of its interrupt processing, the user's error routine can also use this index register without saving and restoring it. However, the user cannot depend on the contents of XR1 unless he initializes it as part of his error routine.

Programming Techniques - Error Routine Exits. Some programming techniques that can be used in conjunction with the ISS error exit follow:

1. To try the operation again:

```
USER    DC      0
        BSC I   USER
```

2. To terminate the operation:

```
USER    DC      0
        SRA     16      (to clear the accumulator)
        BSC I   USER
```

3. To indicate that a condition ("last card" or "channel 9") was detected and that the normal program flow should be altered:

```
        LD      INDIC
        BSC L   NEW, Z (alter program flow)
        LIBF    CARD1
        DC      /1000
        DC      INPUT
        DC      USER
        .       .
        .       .
        .       .
USER    DC      0
        BSC I   USER, Z
        LD      D0001
        STO     INDIC
EXIT    BSC I   USER
        .       .
        .       .
        .       .
NEW     SRA     16
        STO     INDIC
        .       .
        .       .
        .       .
```

BASIC ISS CALLING SEQUENCE

Each ISS described in this manual is entered via a calling sequence. These calling sequences follow a basic pattern. In order not to burden the reader with redundant descriptions, this section presents the basic calling sequences and describes those parameters which are common to most of the subroutines.

Basic Calling Sequence

```
LIBF    Name
DC      Control parameter
DC      I/O area
DC      Error routine
```

The above calling sequence, with the parameters shown, is basic to most of the ISS. Detailed descriptions of the above four parameters are omitted when the subroutines are described later in the manual. Unless otherwise specified, the subroutine returns control to the instruction immediately following the last parameter.

Name Parameter

Each subroutine has a symbolic name, which must be written in the LIBF statement exactly as listed in Table 1 because the object program loader must recognize the name to generate the proper linkage.

Table 1. ISS Names

Subroutine	Name
Card Reader Punch	CARD0 or CARD1
Disk	DISK0 or DISK1 or DISKN
Printer	PRNT1
Console Printer - Input Keyboard	TYPE0
Console Printer	WRTY0
Paper Tape	PAPT1 or PAPTN
Plotter	PLOT1

For some devices multiple subroutines are available, although only one can be selected for use in any program (including called subroutines).

NAME0. The NAME0 subroutine is the shortest and least complicated. The NAME0 version is the standard routine for the card read-punch and console printer-input keyboard. The NAME0 version of the Disk routine can be used if transfer of data is 320 words, or less.

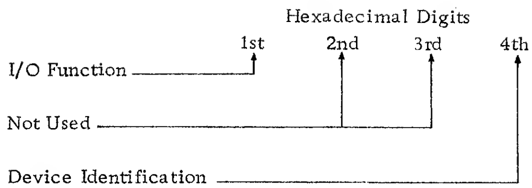
NAME1. The NAME1 version is the standard routine for the disk, 1132 printer, paper tape, and plotter. It may be used for the card read-punch if a user error exit is needed rather than the internal looping and retries by the CARD0 routine.

NAMEN. The NAMEN version is available to operate the paper tape reader and punch simultaneously and to minimize extra disk revolutions when transferring more than 320 words to/from the disk. The NAMEN subroutine is more extensive than the NAME1 subroutine.

Control Parameter

The control parameter, in the form of four hexadecimal digits, conveys necessary control data to the ISS by specifying the desired function (read, write, etc.), the device identification, and similar control information. Most subroutines do not use all four digits.

A typical control parameter is illustrated below.



Since the I/O function and device identification are used in most subroutines, a description of the purpose of each is given here.

I/O Function

The function digit in the calling sequence specifies which I/O operation the user is requesting. Three of these functions, read, write, and test are used in most subroutines.

Read. The read function causes a specified amount of data to be read from an input device and placed in a specified input area. Depending upon the device, an interrupt signals the subroutine either when the next character is ready or when all requested data has been read. When the specified number of characters has been read, the subroutine becomes available for another call to that device.

Write. The write function causes a specified amount of data from the user's output area to be written (or punched) by an output device. As with the read function, an interrupt signals the subroutine when the device can accept another character, or when all characters have been written. When the specified number of characters has been written, the subroutine becomes available for another call to that device.

Test. The test function causes a check to be made as to the status of a previous operation by that subroutine. If the previous operation has been completed, the subroutine branches to the LIBF +3 core location; if the previous operation has not been completed, the subroutine branches to the LIBF +2 core location. The test function is illustrated below:

```
          LIBF        Name
LIBF +1   DC          Control Parameter
                        (specifying Test function)
LIBF +2   OP Code     xxxx....
LIBF +3   OP Code     xxxx....
```

NOTE: Specifying the test function requires two statements (one LIBF and one DC), except in Disk subroutines, where three statements are required.

This function is useful in situations where input data has been requested, and no processing can be done until that data is available.

Device Identification

This digit should be zero except for the Test function with the PAPTN (paper tape) subroutine.

NOTE: For all disk subroutines, this digit appears in the I/O area rather than in the control parameter.

I/O Area Parameter

The I/O area for a particular operation consists of one table of control information and data. This table is composed of a data area preceded by a control word (two control words for disk operations) that specifies how much data is to be transferred. The area parameter in the calling sequence is the address (symbolic or actual) of the first control word that precedes the data area.

The control word contains a word count referring to the number of data words in the table. It is important to remember that the number of words in the table is not always the number of characters to be read (or written) because some codes pack several characters per word. The disk subroutines require a second control word, which is described along with those subroutines.

Error Parameter

The error parameter is the means by which an ISS can give temporary control to the user in the event of conditions such as error, last card, etc. This parameter is not required for the NAME0 subroutines for the 1442 or the Console Printer or Input-Keyboard. The instruction sequence for setting up the error routine is shown below.

```
        LIBF    NAME
         .       .
        DC      ERROR (error parameter)
         .       .
ERROR   BSS     1        (return link)
         .       .
         .       .       (error routine)
        BSC I   ERROR (branch to return link)
```

The return link is the address in the related ISS to which control must be returned upon completion of the error routine. The link is inserted in location ERROR by a BSI from the ISS when the subroutine branches to the error routine.

The types of errors that cause a branch to the error address are listed in Appendix B.

NOTE: The user error routine is executed as part of the interrupt response handling. The interrupt level is still on and remains on until control is returned to the ISS (see General Error Handling Procedures).

ASSIGNMENT OF CORE STORAGE LOCATIONS

The portion of core storage used by the ISS and ILS subroutines is defined below. Care should be used in altering any of these locations (see Figure 3).

The areas called out in Figure 3 are described below.

Interrupt Branch Addresses

ILS Routines. The ILS00 routine is always assigned to location 8, ILS01 to location 9, . . . , ILS05 to location 13.

Interrupt Trap. The address of the interrupt routine trap is stored in any location for which no ILS routine is loaded.

1132 Printer

This area is used by 1132 Printer.

ISS Error Exit

This exit is used whenever a preoperation error (illegal LIBF or device not ready) is detected by an ISS.

To retry the call, push START.

ISS Exit

The ISS exit results from a keyboard operator request.

The TYPE0 and WRTY0 subroutines execute a BSI I 44 whenever a keyboard operator request is detected. Note that interrupt level 04 is still on.

Hex	Decimal		
8	8	(ILS 00)	Interrupt Branch Addresses
9	9	(ILS 01)	
A	10	(ILS 02)	
B	11	(ILS 03)	
C	12	(ILS 04)	
D	13	(ILS 05)	
E	14		Unused
1F	31		
20	32		Reserved for 1132 Printer
27	39		
28	40	DC 0	ISS Error Exit
29	41	WAIT	
		BSC I 40	
2C	44	DC 45	ISS Exit (Keyboard Operator Request)
2D	45	DC 0	Interrupt Trap
2E	46	WAIT	
2F	47	MDX *-2	
		BOSC I 45	
32	50	DC 0	ISS Counter

Figure 3. ISS and ILS Core Locations

The user-written subroutine must return to the TYPE0 or WRTY0 subroutine in order to allow interrupts of equal or lower priority to occur. Also a call executed to any subroutine might cause a recurrent-entry problem unless the user can guarantee that the subroutine was not in use when the keyboard interrupt occurred.

This location (44) is initialized for the interrupt trap by the relocating loader, in case the user fails to store an address in the interrupt trap to process keyboard operator requests.

Interrupt Trap

This routine is entered when an interrupt occurs for which there is no processing routine, e.g., no ILS routine loaded, no ISS routine assigned to the pertinent ILSW bit.

Interrupts of higher priority will be processed before the computer finally halts with the IAR displaying 002A.

ISS Counter

The ISS eounter is inerernented by +1 every time an ISS initiates an interrupt-causing I/O operation and deerernented by +1 when the operation is complete. A non-zero content indieates interrupt(s) pending.

DESCRIPTIONS OF INTERRUPT SERVICE SUBROUTINES

CARD SUBROUTINES

The card subroutines perform all I/O functions relative to the IBM 1442 Card Read Punch; viz., read, punch, feed, and select staeker.

CARD0 Subroutine. The CARD0 subroutine is shorter and less conlplieated and is the standard routine for the Card Read Puneh.

The CARD0 subroutine ean be used if the error parameter is not needed. On an error, the subroutine loops, waiting for operator intervention; last card eonditions cause pre-operative not ready exits.

CARD1 Subroutine. The CARD1 subroutine can be used for the Card Read Punch if a user error exit is needed, rather than the internal looping and retries of the CARD0 routine.

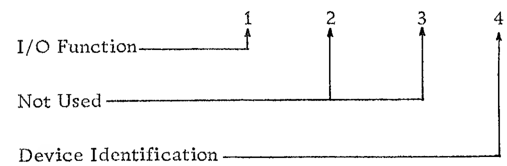
Calling Sequence

```
        LIBF          CARD0 or CARD1
        DC            /xxx0 (Control)
        DC            AREA (I/O area)
        DC            ERROR (Error routine)
         :               :
ERROR  | Return Link              |
       | Error Routine            |
       |                          |
       | BSC I              ERROR |
         .               .
         .               .
AREA   | Word Count               |
       | I/O Area                 |
       |                          |
```

The calling sequenee parameters are described in the following paragraphs.

Control Parameter

This parameter eonsists of four hexadeeimal digits as shown below:



I/O Funetion

The I/O function digit specifies a particular operation performed on the 1442 Card Read Puneh. The funetions, associated digital values, and required parameters are listed and deseribed below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read	1	Control, I/O Area, Error**
Punch	2	Control, I/O Area, Error**
Feed	3	Control, Error**
Select Stacker	4	Control

* Any parameter not required for a particular function must be omitted.
**Error parameter not required for CARD0.

Test. Branches to LIBF+2 if the previous operation has not been eompleted, to LIBF+3 if the previous operation has been completed.

Read. Reads one card and transfers a speeified number of columns of data to the user's input area. The number of eolumns read (1-80) is specified by the user in the first loeation of the I/O area. The subroutine clears the remainder of the I/O area and stores a 1 in bit position 15 of each word, initiates the eard operation, and returns control to the user's program. When each eolumn is ready to be read, a eolumn interrupt occurs. This permits the card subroutine to read the data from that eolumn into the user's input area (clearing bit 15), after which the user's program is again resumed. This sequence of events is repeated until the requested number of eolumns has been read, after which the remaining column interrupts are cleared (no data read).

When an operation complete interrupt occurs, the card subroutine checks for errors, informs the user if an error occurred (CARD1 only), and sets up to terminate (CARD1 only) or retry the operation.

The data in the user's input area is in card code format; that is, each 12-bit column image is left-justified in one 16-bit word.

Punch. Punches into one card the number of columns of data specified by the word count found at the beginning of the user's output area. The punch operation is similar to the read operation. As each column comes under the punch dies, a column interrupt occurs; the card subroutine transfers a word from the user's output area to the punch and then returns control to the user's program.

This sequence is repeated until the requested number of columns has been punched, after which an Operation Complete interrupt occurs. At this time the card subroutine checks for errors, informs the user if an error occurred (CARD1 only), and sets up to terminate (CARD1 only) or retry the operation. The character punched is the image of the leftmost 12 bits in the word.

Feed. Initiates a card feed cycle. This advances all cards in the machine to the next station, i.e., a card at the punch station advances to the stacker, a card at the read station advances to the punch station, and a card in the hopper advances to the read station. No data is read or punched as a result of a feed operation and no column interrupts occur.

When the card advance is complete, an Operation Complete interrupt occurs. At this time the card subroutine checks for errors, informs the user if an error occurred (CARD1 only), and sets up to terminate (CARD1 only) or retry the operation.

Select Stacker. Selects stacker 2 for the card currently at the punch station. After the card passes the punch station, it is directed to stacker 2.

Device Identification

This digit must be zero.

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. The control word consists of a word count that specifies the number of columns of data read or punched, always starting with column 1.

Error Parameter

CARD0. CARD0 has no error parameter. If an error is detected while an Operation Complete interrupt is being processed, the subroutine loops on not ready, with interrupt level 4 on, waiting for operator intervention. When the condition has been corrected and the 1442 made ready, the subroutine attempts the operation again.

CARD1. CARD1 has an error parameter. If an error is detected, the user can request the subroutine to terminate (clear routine-busy indicator and the interrupt level) or to loop on not ready waiting for operator intervention (interrupt level 4 on). (See Basic Calling Sequence.)

Protection of Input Data

Since the CARD subroutines read data directly into the user's I/O area, the user can manipulate the data before the entire card has been processed. This procedure is inherently dangerous because, if an error occurs, the data may be in error and error recovery procedures will cause the operation to be tried again. The exit via the error parameter is the only method of informing the user that an error has occurred. Therefore, do not manipulate data before the entire card has been processed when using CARD0.

When using CARD1, the following precautions should be taken:

- Do not store converted data back into the read-in area.
- Do not take any irretrievable action based on the data until the card has been read correctly; i.e., be prepared to convert the data or perform the calculations a second time.
- When data manipulation is complete, check the user-assigned error indicator that is set when a branch to the user-written error routine occurs. The data conversion or calculations can then be reinitiated, if necessary.

Last Card

A read or feed function requested after the last card has been detected will eject that card and cause a branch to the pre-operative error exit (location 41). A punch function will punch and then eject that card with a normal exit. Therefore, to eject the last card without causing a pre-operative error exit, request a punch function with a word count of one and a blank in the data field.

DISK SUBROUTINES

The disk subroutines perform all reading and writing of data relative to Disk Storage. This includes the major functions: seek, read, and write, in conjunction with readback check, file-protection, and defective sector handling.

DISK0. The DISK0 subroutine is the shortest and least complicated and can be used if not more than 320 words are to be read or written at one time.

DISK1. The DISK1 version is the standard routine for the Disk and allows consecutive sectors to be read or written; however, a full disk revolution might occur between sectors.

DISKN. The DISKN subroutine minimizes extra disk revolutions in transferring more than 320 words. The DISKN subroutine is more extensive than DISK1.

One of the major differences among the disk subroutines is the ability to read or write consecutive sectors on the disk without taking an extra revolution. If a full sector is written, the time in which the I/O command must be given varies. DISKN is programmed so that it can "make" the sector gap the majority of the time; DISK1 approximately 50 percent of the time; and DISK0 (if LIBF's follow one another closely) only on a Read or Write Immediate function, since both Write functions require reading of the sector address to verify the arm positioning.

All three disk subroutines have the same error handling procedures.

NOTE: In the 1130 Monitor System,the disk subroutines are a part of the supervisor and as such are not loaded with the subroutine library. Consequently, these routines do not have LET entries.

Sector Numbering and File Protection

In the interest of providing disk features permitting versatile and orderly control of disk operations, three important conventions have been adopted. They are concerned with sector numbering, file protection, and defective sector handling. Successful use of the disk subroutines can be expected only if user programs are built within the framework of these conventions.

The primary concern behind the conventions is the safety of data recorded on the disk. To this end, the file-protection scheme plays a major role, but does so in a manner that is dependent upon the sector-numbering technique. The latter contributes to data safety by allowing the disk subroutine to verify the correct positioning of the access arm before it actually performs a Write operation. This verification requires that sector identifications be prerecorded on each sector and that subsequent writing to the disk be done in a manner that preserves the existing identification. The disk subroutines have been organized to comply with these requirements.

Sector Numbering

The details of the numbering scheme are as follows: each disk sector is assigned an address from the sequence 0, 1, . . . , 1623, corresponding to the sector position in the ascending sequence of cylinder and sector numbers from cylinder 0 (outer-most) sector 0, through cylinder 202 (inner-most) sector 7. (The user can address cylinders 0 through 199. The remaining three cylinders are reserved for defective-sector handling. Each cylinder contains eight sectors and each sector contains 321 words.) The sector address is recorded in the first word of each sector and occupies the rightmost eleven bit positions. Of these eleven positions, the three low-order positions identify the sector (0-7) within the cylinder. Utilization of this first word for identification purposes reduces the per sector availability of data words to 320; therefore, transmission of full sectors of data is performed in units of this amount. The sector addresses must be initially recorded on the disk by the user and are thereafter rewritten by the disk subroutines as each sector is written.

File Protection

File protection is provided to guard against the inadvertent destruction of previously recorded data. By having the normal writing functions uniformly test for the file-protection status of sectors they are about to write, this control can be achieved.

This is implemented by assigning a file-protected area for each disk. The address of the first unprotected sector (0000-1623) on each disk is stored within the Disk subroutine. Every sector below this one is file-protected. In the Disk Monitor System, assignment is made by the system. In the Card/Paper Tape System, the user must make the assignment.

Defective Sector Handling

A defective sector is one in which, after ten retries, a successful writing operation cannot be completed. A cylinder having one or more defective sectors is defined as a defective cylinder. The disk subroutines can operate when as many as three cylinders are defective.

Since there are 203 cylinders on each disk, the subroutine can "overflow" the normally used 200 cylinders when defective cylinders are encountered (see Effective Address Calculation).

The address of each defective cylinder is stored within the Disk subroutines (Card/Paper Tape system) or in COMMA (Monitor System). In the Card/Paper

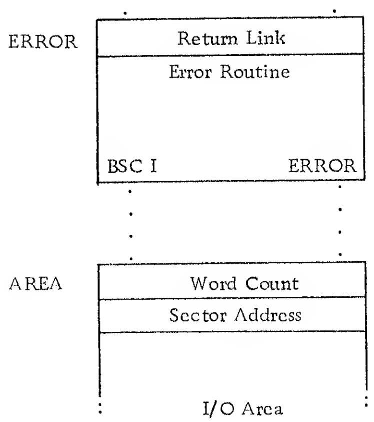
Tape System, these addresses must be stored by the user (see Disk Initialization).

If a cylinder becomes defective during an operation, the user can move the data in that cylinder and each higher-addressed cylinder into the next higher-addressed cylinders. Then the address of the new defective cylinder can be stored in DISKx +16, +17, or +18 and normal operation continued.

The user should not store the new defective cylinder address in DISKx and then continue normally because the effective sector address computation then yields a sector address eight higher than is desired (see Effective Address Calculation).

Calling Sequence

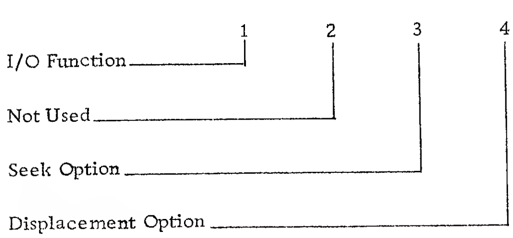
```
        LIBF        DISK0 or
                    DISK1 or DISKN
        DC          /xxxx        (Control)
        DC          AREA         (I/O area)
        DC          ERROR        (Error routine)
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, shown below:



I/O Function

The I/O function digit specifies the operation to be performed on Disk Storage. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control, I/O Area
Read	1	Control, I/O Area, Error
Write without RBC	2	Control, I/O Area, Error
Write with RBC	3	Control, I/O Area, Error
Write Immediate	4	Control, I/O Area
Seek	5	Control, I/O Area, Error

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF +3 if the previous operation has not been completed, to LIBF +4 if the previous operation has been completed.

NOTE: This function requires two parameters

Read. Positions the access arm and reads data into the user's I/O area until the specified number of words has been transmitted. Although sector identification words are read and checked for agreement with expected values, they are neither transmitted to the I/O data area nor are they counted in the tally of words conveyed.

If, during the reading of a sector, a read check occurs, up to ten retries are attempted. If the error persists, the function is temporarily discontinued, an error code is placed in the accumulator, the address of the faulty sector is placed in the extension, and an exit is made to the error routine specified by the error parameter.

Upon return from the error routine, that sector operation is reinitiated or the function is terminated, depending on whether the accumulator is non-zero or zero.

Write With Readback Check. This function first checks whether or not the specified sector address is in a file-protected area. If it is, the subroutine places the appropriate error code in the accumulator and exits to location 41.

If the specified sector address is not in a file-protected area, the subroutine positions the access arm and writes the contents of the indicated I/O data area into consecutive disk sectors. Writing

begins at the designated sector and continues until the specified number of words have been transmitted. A readback check is performed on the data written.

If any errors are detected, the operation is retried up to ten times. If the function cannot be accomplished by this time, an appropriate error code is placed in the accumulator, the address of the faulty sector is placed in the extension, and exit is made to the error routine designated by the Error parameter.

Upon return from this error routine, that sector operation is reinitiated or the function is terminated depending upon whether the accumulator is non-zero or zero.

As each sector is written, the subroutine supplies the sector identification word. The identification word for the first sector is obtained from the I/O area, although it and subsequently generated identification words are not included in the word count.

Write Without Readback Check. This function is the same as the function described above except that no readback check is performed.

Write Immediate. Writes data with no attempt to position the access arm, check for file-protect status, or check for errors. Writing begins at the sector number specified by the rightmost three bits of the sector address. This function is provided to fulfill the need for more rapid writing to the disk than is provided in the previously described Write functions. Primary application will be found in the "streaming" of data to the disk for temporary bulk storage.

As each sector is written, the subroutine supplies the sector identification word. The identification word for the first sector is obtained from the I/O area, although it and subsequently generated identification words are not included in the word count.

Seek - Initiates a seek as specified by the seek option digit. If any errors are detected, the operation is tried again up to ten times.

Seek Option

If zero, a seek is executed to the cylinder whose sector address is in the disk I/O area control word; if non-zero, a seek is executed to the next cylinder toward the center, regardless of the sector address in the disk I/O area control word. This option is valid only when the seek function is specified.

The seek function requires that the user set up the normal I/O area parameter (see I/O Area Parameter) even though only the sector address in the I/O area is used. The I/O area control (first) word is ignored.

Displacement Option

If zero, the sector address word contains the absolute sector identification; if non-zero, the file protect address for the specified disk is added to bits 4-15 of the sector address word to generate the effective sector identification. The file-protect address is the sector identification of the first unprotected sector.

I/O Area Parameter

The I/O area parameter is the label of the first of two control words which precede the user's I/O area. The first word contains a count of the number of data words that are to be transmitted during the disk operation.

If the DISK1 or DISKN subroutine is used, this count need not be limited by sector or cylinder size, since the subroutines cross sector and cylinder boundaries, if necessary, in order to process the specified number of words. However, if the DISK0 subroutine is used, the count is limited to 320.

The second word contains the sector address where reading or writing is to begin. Bits 0-3 are the device identification and must be zero. Bits 4-15 specify the sector address. Following the two control words is the user's data area.

Error Parameter

Refer to the section, Basic Calling Sequence.

Important Locations

The relative locations within the DISK0, DISK1, and DISKN subroutines are defined as follows:

DISKx +0 - entry point from calling transfer vector when LIBF DISKx is executed.
+2 - loader stores address of first location (in the calling transfer vector) assigned to DISKx
+4 - entrance from ILS subroutine handling Disk Storage interrupts.
+7 - area code for disk storage.
+8 - zero
+9 - zero

+10 - cylinder identification (bits 4-12) of the cylinder currently under the disk read/write heads (loaded as +202)
+11 - unused
+12 - unused
+13 - sector address (bits 4-15) of the first non-file-protected sector for Disk Storage (loaded as 0)
+14 - unused
+15 - unused
+16 - sector address of the first defective cylinder for Disk Storage (loaded as +1624)
+17 - sector address of the second defective cylinder for Disk Storage (loaded as +1624)
+18 - sector address of the third defective cylinder for Disk Storage (loaded as +1624)

In the disk monitor system, words DISKx +10 through DISKx +18 are stored in COMMA.

Effective Address Calculation

Effective address calculation is as follows:

1. Start with the user-requested sector address (found in the sector address word of the I/O area).
2. If the displacement option (found in the control parameter) is non-zero, add in the sector address of the first non-file-protected sector (found in DISKx +13).

   NOTE: This starting address will cause a pre-operative error exit to location 41 if over +1599.
3. If the resulting address is equal to or greater than the sector address of the first defective cylinder (found in DISKx +16), add +8.
4. If the resulting address is equal to or greater than that of the second defective cylinder (found in DISKx +17), add +8.
5. If the resulting address is equal to or greater than that of the third defective cylinder (found in DISKx +18), add +8.
6. The resulting address is the effective sector address.

Disk Initialization

It is the card/paper tape system user's responsibility to correctly load DISKx +13, +16, +17, and +18 at execution time and whenever a new disk pack is inserted. The following routines can be used to accomplish this.

Disk Pack Initialization Routine (DPIR). The functions of this routine are to write sector addresses on a disk pack, to detect any defective cylinders, and to store defective cylinder and file protect information and a disk pack label in sector 0 of the disk pack, see IBM 1130 Card/Paper Tape Programming Systems Operating Guide (Form C26-3629).

Set Pack Initialization Routine (SPIR0, SPIR1, and SPIRN). The function of this routine is to store defective cylinder and file protect information from sector 0 of the disk pack into the appropriate DISKx subroutine.

If the above routines are not used, the starting address of the DISKx routine can be loaded into an index register for easy use in reaching the specified locations:

```
        LD        LIBF
        SLA       8         expand modifier into
        SRT       8         16 bits with sign
        STX   3   LOAD+1
        A         LOAD+1    add in TV address
        A         D0002     add constant to reach 3rd
        STO       LOAD+1    word of DISKx slot
LOAD    LDX   I2  0         XR2 = DISKx
        .         .
        .         .
D0002   DC        +2
LIBF    BSI   3   n         source = LIBF DISKx
        .         .
        .         .
        .         .
c(XR3)+n DC       0         loaded as calling
        BSC   L   DISKx     transfer sector (TV)
```

SET PACK INITIALIZATION

The SPIR is a special-purpose utility routine, available to the Card/Paper Tape System user. It is not called by LIBF as are the other Disk subroutines described in this section. SPIR0 must be used if DISK0 is called, SPIR1 if DISK1 is called, or SPIRN if DISKN is called.

NOTE: In no case should SPIR be used in the monitor system.

The SPIR reads sector 0000 from the disk and stores the first four words into the disk ISS that is in core. This routine should be called before any calls are made to the disk ISS.

The calling sequence for SPIR is as follows:

```
CALL      SPIRn

DC        /0000
```

The four words read from sector 0000 are described under Disk Pack Initialization Routine. See the publication IBM 1130 Card/Paper Tape Programming System Operator's Guide (Form C26-3629).

The information was stored on the disk by the DPIR.

PRINTER SUBROUTINES

The printer subroutine PRNT1 handles all print and carriage control functions relative to the IBM 1132 Printer. Only one line of data can be printed, or one carriage operation executed, with each call to the printer subroutine. The data in the output area must be in EBCDIC form, packed two characters per computer word. (See Data Codes.)

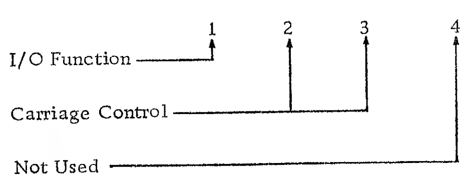
Calling Sequence

```
            LIBF          PRNT1
            DC            /xxxx   (control)
            DC            AREA    (I/O area)
            DC            ERROR   (Error routine)
             .              .
             .              .
             .              .
ERROR    +---------------------------------+
         |          Return Link            |
         +---------------------------------+
         |         Error Routine           |
         |                                 |
         |                                 |
         | BSC I                   ERROR   |
         +---------------------------------+
             .              .
             .              .
             .              .
AREA     +---------------------------------+
         |          Word Count             |
         +---------------------------------+
         |           I/O Area              |
         |                                 |
         |                                 |
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below.



I/O Function

The I/O function digit specifies the operation to be performed on an 1132 Printer. The functions, their associated digital values, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Print	2	Control, I/O Area, Error
Control Carriage	3	Control
Print Numerical	4	Control, I/O Area, Error

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF+2 if the previous operation has not been completed or to LIBF+3 if the previous operation has been completed.

Print. Prints characters from the user's I/O area, checking for channel 9 and 12 indications. If either of these conditions is detected, the subroutine branches to the user's error routine after the line of data has been printed. Upon return from this error routine, a skip to channel 1 is initiated or the function is terminated, depending upon whether the Accumulator is non-zero or zero.

Control Carriage. Controls the carriage as specified by the carriage control digits listed in Table 2.

Print Numerical. Prints only numerals and special characters from the user's I/O area and checks for channel 9 and channel 12 indications. See Print above.

Carriage Control

Digits 2 and 3 specify the carriage control functions listed in Table 2. An immediate request is executed before the next print operation; an after-print request is executed after the next print operation and replaces the normal space operation.

If the I/O function is print, only digit 3 is examined; if the I/O function is control, and digits 2 and 3 both specify carriage operations, only digit 2 is used.

Table 2. Carriage Control Operations

Digit #2: Immediate Carriage Operations
Print Functions Not Used
Control Function 1 - Immediate Skip To Channel 1 2 - Immediate Skip To Channel 2 3 - Immediate Skip To Channel 3 4 - Immediate Skip To Channel 4 5 - Immediate Skip To Channel 5 6 - Immediate Skip To Channel 6 9 - Immediate Skip To Channel 9 C - Immediate Skip To Channel 12 D - Immediate Space Of 1 E - Immediate Space Of 2 F - Immediate Space Of 3
Digit #3: After-Print Carriage Operations
Print Functions 0 - Space One Line After Printing 1 - Suppress Space After Printing
Control Function 1 - Skip After Print To Channel 1 2 - Skip After Print To Channel 2 3 - Skip After Print To Channel 3 4 - Skip After Print To Channel 4 5 - Skip After Print To Channel 5 6 - Skip After Print To Channel 6 9 - Skip After Print To Channel 9 C - Skip After Print To Channel 12 D - Space 1 After Print E - Space 2 After Print F - Space 3 After Print

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. The control word consists of a word count that specifies the number of computer words of data to be printed. The data must be in EBCDIC format, packed two characters per computer word.

Error Parameter

See Basic Calling Sequence.

CONSOLE PRINTER/INPUT KEYBOARD

There are two ISS for the transfer of data to and from the Console Printer and the Input Keyboard.

TYPE0. The TYPE0 subroutine handles input and output.

WRTY0. The WRTY0 subroutine handles output only. If a program does not require keyboard input, it is advantageous to use the WRTY0 subroutine because it occupies less core storage than the TYPE0 subroutine.

Only the TYPE0 subroutine is described below; the WRTY0 subroutine is identical, except that it does not allow the Read-Print function.

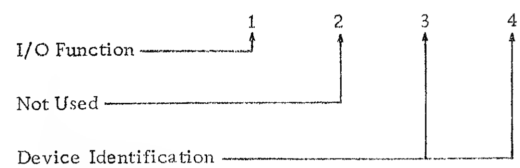
Calling Sequence

```
          LIBF        TYPE0 or WRTY0
          DC          /xxxx  (Control)
          DC          AREA   (I/O area
           .            .
           .            .
AREA      | Word Count |
          | I/O Area   |
          |            |
```

The parameters used in the above calling sequence are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specifies the operation to be performed on the input keyboard and/or console printer. The functions, their associated digital values, and the required parameters are listed and then described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read-Print	1	Control, I/O Area
Print	2	Control, I/O Area

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF+2 if the previous operation has not been completed or to LIBF+3 if the previous operation has been completed.

Read-Print. Reads from the keyboard and prints the requested number of characters on the console printer. The operation sequence is as follows:

1. The calling sequence is analyzed by the Call routine, which then unlocks the keyboard.
2. When a key is pressed, a character interrupt signals the Interrupt Response Routine that a character is ready to be read into core storage.
3. The Interrupt Response Routine converts the keyboard data to console printer output code (see Data Codes). Each character is printed as it is read; the keyboard is then unlocked for entry of the next character.
4. Printer interrupts occur whenever the console printer has completed a print operation. When the interrupt is received, the routine checks to determine if the final character has been read and printed. If so, the operation is considered complete. If the console printer becomes not ready during printing, the subroutines loop internally, waiting for the console printer to become ready.
5. Steps 2 to 4 are repeated until the specified number of characters have been read and printed. The characters read into the I/O area are in IBM card code; that is, each 12-bit image is left-justified in one 16-bit word.

Print. Prints the specified number of characters on the console printer. A printer interrupt occurs when the console printer has completed a print operation. When an interrupt is received, the character count is checked. If the specified number of characters has not been written, printing is initiated for the next character. This sequence continues until the specified number of characters has been printed. Data to be printed must be in console printer code, (see Data Codes) packed two characters per 16-bit word. Control characters can be embedded in the message where desired.

In Read-Print and Print operations, printing begins where the printing element is positioned; that is, carrier return to a new line is not automatic when the subroutine is called.

Device Identification

Device identification digits can be 00 or 01; either value specifies the console printer.

Keyboard Functions

Keyboard functions provide for control by the TYPE0 subroutine and by the operator.

TYPE0 Subroutine Control

Three keyboard functions are recognized by the TYPE0 subroutine.

Backspace. The operator presses the backspace key whenever the previous character is in error. The interrupt response routine senses the control character, backspaces the console printer, and prints a slash (/) through the character in error. In addition, the subroutine prepares to replace the incorrect character in the I/O area with the next character.

If the backspace is depressed twice, the character address is decremented by +2, but only the last graphic character is slashed. For example, if ABCDE was entered and then the backspace key depressed three times, the next graphic character replaces the C but only the E is slashed each time. If XYZ is the new entry, the print-out shows ABCD~~E~~XYZ, but the buffer contains ABXYZ.

Erase Field. When the interrupt response routine recognizes the erase field control character, it assumes that the entire message is in error and is to be entered again. The routine prints two slashes on the console printer, restores the carrier to a new line, and prepares to replace the old message in the I/O area with the new message.

The old message in the I/O area is not cleared. Instead, the new message overlays the old, character by character. If the old message is longer than the new, the remainder of the old message follows the NL character terminating the new message.

End-of-Message. When the interrupt response routine recognizes the end-of-message control character, it assumes the message has been completed, stores an NL character in the I/O area, and terminates the operation.

Operator Request Function

By pressing the operator request key on the keyboard, the operator can inform the program that he wishes to enter data from the keyboard or the Console Entry switches. The interrupt that results causes the console printer routine to execute an indirect BSI instruction to core location 44, where the user must have the address of an operator request routine stored. Bit 1 of the accumulator contains the keyboard/console identification bit; that is, the device status word, shifted left two bits.

The user's operator request routine must return to the ISS subroutine via the return link. The user's routine is executed as a part of the interrupt handling. The interrupt level remains ON until control is returned to the ISS subroutine (see General Error Handling Procedures, Post-operation Checks).

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. The control word consists of a word count that specifies the number of words to be read or printed. This word count is equal to the number of characters if the Read-Print function is requested, but to one-half the number of characters if the Print function is requested.

PAPER TAPE SUBROUTINES

The paper tape subroutines handle the transfer of data from a Paper Tape Reader to core storage and from core storage to a Paper Tape Punch. Any number of characters can be transferred via one calling sequence.

The PAPTN subroutine must be used if simultaneous reading and punching are desired.

The PAPT1 operates both devices, but only one at a time.

When called, the paper tape subroutine starts the reader or punch and then, as interrupts occur, transfers data to or from the user's I/O area. Input data is packed two characters per computer word by the subroutine; output data must be in that form when the subroutine is called for a punch function.

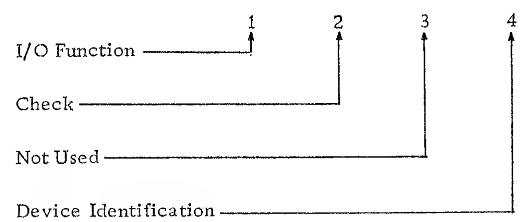
Calling Sequence

```
        LIBF        PAPT1 or PAPTN
        DC          /xxxx  (Control)
        DC          AREA  (I/O area)
        DC          ERROR  (Character)
         .             .
         .             .
ERROR  |      Return Link      |
       |     Error Routine     |
       |                       |
       | BSC I           ERROR |
         .             .
         .             .
         .             .
AREA   |      Word Count       |
       |       I/O Area        |
       |                       |
```

The parameters used in the above calling sequence are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specifies the operation to be performed on a Paper Tape Attachment. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameter*
Test	0	Control
Read	1	Control, I/O area, Error
Punch	2	Control, I/O area, Error

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF+2 if the previous operation has not been completed or to LIBF+3 if the previous operation has been completed.

Read. Reads paper tape characters into the specified number of words in the I/O area. Initiating reader motion causes an interrupt to occur when a character can be read into core. If the specified number of words has not been filled, or the stop character has not been read (see Check), reader motion is again initiated.

Punch. Punches paper tape characters into the tape from the words in the I/O area. Each character punched causes an interrupt when the next character can be accepted. The operation is terminated by transferring either a stop character or the specified number of words.

Check

The check digit specifies whether or not word-count checking is desired while completing a read or punch operation as shown below:

0 Check
1 No check

Check. This function should be used with the Perforated Tape and Transmission Code (PTTC/8) only (see Data Codes). The PTTC/8 code for DEL is used as the delete character when reading. The delete character is not placed in the I/O area and therefore does not enter into the count of the total number of words to be filled.

The PTTC/8 code for NL is used as the stop character when doing a Read or Punch. On a Read operation, the NL character is transferred into the I/O area. On a Punch operation, the NL character is punched into the paper tape.

When the NL character is encountered before the specified number of words has been read or punched, the operation is terminated. When the specified number of words has been read or punched, the operation is terminated, even though a NL character has not been encountered.

No Check. The Read or Punch function is terminated when the specified number of words has been read or punched. No checking is done for a DEL or NL character.

Device Identification

When the Test function is specified, the PAPTN subroutine must be told which device (reader or punch) is to be tested for an operation complete indication. (Remember that both the reader and the punch can operate simultaneously.) Therefore, the device identification is used only for the Test function in the PAPTN subroutine. If device identification is a 0, the subroutine tests for a reader complete indication; if the code is a 1, the subroutine tests for a punch complete indication.

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area and consists of a word count that specifies the number of words to be read into or punched from core. Since characters are packed two per word in the I/O area, this count is one-half the maximum number of characters transferred. Because an entire eight-bit channel image is transferred by the subroutine, any combination of channel punches is acceptable. The data can be a binary value or a character code. The code most often used is the PTTC/8 code. (See Data Codes.)

Error Parameter

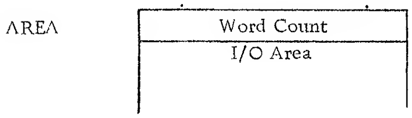
See Basic Calling Sequence.

PLOTTER SUBROUTINES

The plotter subroutine converts hexadecimal digits in the user's output area into actuating signals that control the movement of the plotter recording pen. Each hexadecimal digit in the output area is translated into a plotter operation that draws a line segment or raises or lowers the recording pen. The amount of data that can be recorded with one calling sequence is limited only by the size of the corresponding output area.

Calling Sequence

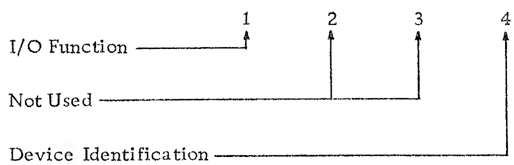
```
        LIBF        PLOT1
        DC          /xxxx   (Control)
        DC          AREA    (I/O area)
        DC          xxxx    (Error routine)
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specifies the operation to be performed on the Plotter. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameter*
Test	0	Control
Write	1	Control, I/O Area, Error

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF+2 if the previous operation has not been completed or to LIBF+3 if the previous operation has been completed.

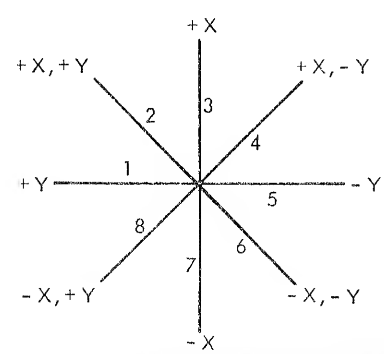
Write. Transforms hexadecimal digits in the output area into signals that actuate the plotter. Table 3 lists the hexadecimal digits and the plotting actions they represent. Figure 4 shows the binary and hexadecimal configurations for drawing the letter E.

Device Identification

This digit must be zero.

Table 3. Plotter Control Digits

Hexadecimal Digit	Plotter Action (See Diagram Below)
0	Pen Down
1	Line Segment = +Y
2	Line Segment = +X, +Y
3	Line Segment = +X
4	Line Segment = +X, -Y
5	Line Segment = -Y
6	Line Segment = -X, -Y
7	Line Segment = -X
8	Line Segment = -X, +Y
9	Pen Up
A	Repeat the previous pen motion the number of times specified by the next digit (Maximum-15 times)
B	Repeat the previous pen motion the number of times specified by the next two digits (Maximum-255 times)
C	Repeat the previous pen motion the number of times specified by the next three digits (Maximum-4095 times)
D	Not Used
E	Not Used
F	Not Used



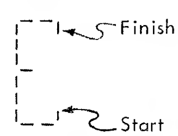
Binary	Hexadecimal	Figure
0000011100010001	0711	
0011101000100101	3A25	
1001000100000011	9103	
1010001001010101	A255	
0111100111111111	79FF	

Figure 4. Plotter Example

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area.

The control word consists of a word count that specifies the number of computer words of data used.

Error Parameter

This parameter is not used but must be included because the routine will return to LIBF+4. (See Basic Calling Sequence.)

INTRODUCTION

Many of the functions and capabilities available within the general I/O and conversion subroutines described in this manual are beyond specification by the FORTRAN language. For example, the FEED function of the 1442 cannot be specified in FORTRAN. Therefore, a set of limited-function I/O and conversion subroutines is included in the subroutine library for use by FORTRAN-compiled programs. Any subroutines written in assembler language that execute I/O operations, and that are intended to be used in conjunction with FORTRAN-compiled programs must employ these special I/O routines for any I/O device specified in a mainline *IOCS record or for any device on the same interrupt level.

The subroutine library contains the following special routines:

CARDZ	-	1442-Input/Output Subroutine
TYPEZ	-	Input Keyboard/Console Printer Input/Output Subroutine
WRTYZ	-	Console Printer Subroutine
PRNTZ	-	1132 Printer Subroutine
PAPTZ	-	Paper Tape Input/Output Subroutine
DISKZ	-	Disk Input/Output Subroutine*
HOLEZ	-	IBM Card Code/EBC Conversion Subroutine
EBCTB	-	EBC/Console Printer Code Table
HOLTB	-	IBM Card Code Table
GETAD	-	Subroutine Used to Locate Start Address of EBCTB/HOLTB

GENERAL SPECIFICATIONS

The FORTRAN I/O device routines operate in a non-overlapped mode. Thus the device routine does not return control to the calling program until the operation is completed.

The input/output buffer for the subroutines is a 121-word buffer starting at location 003C. The maximum amount of data transferable is listed in the description of each subroutine. Output data must be stored in unpacked (one character per word) EBCDIC format, $00XX_{16}$. Data entered from an input device is converted to unpacked (one character per word) EBCDIC format, $00XX_{16}$.

* In the 1130 Monitor System, the disk subroutines are a part of the supervisor and as such are not loaded with the subroutine library. Consequently, these routines do not have LET entries.
DISKZ is not included in the Card/Paper Tape subroutine library.

The EBCDIC character set recognized by the subroutine comprises digits 0-9, alphabetic characters A-Z, blank, and special characters -+.&=(),'/*<%#@. Any other character is recognized as a blank.

The accumulator, accumulator extension, and Index Registers 1 and 2 are used by the FORTRAN device routines and must be saved, if required, before entry into a routine.

The accumulator must be set to zero for input operations.

For output operations, the accumulator must be set to 0002, except for PRNTZ and WRTYZ, in which output is the only valid operation. Index Registers 1 and 2 are set to the number of characters transmitted, except for PRNTZ (1132 Printer) in which Index Register 2 contains the number of characters printed plus an additional character for forms control.

ERROR HANDLING

Device errors, e.g., not ready, read check, result in the execution of a Wait instruction by the routine. After the appropriate corrective action is taken by the operator, PROGRAM START is pressed to execute or reinitiate the operation, as required.

DESCRIPTIONS OF I/O SUBROUTINE

The subroutines described in the sections that follow do not provide a check to determine validity of parameters (contents of accumulator and Index Register 2). Invalid parameters cause indeterminate operation of the subroutines.

TYPEZ KEYBOARD-CONSOLE PRINTER I/O SUBROUTINE

Buffer Size: Maximum of 80 words input, 120 words output.

Keyboard Input: The subroutine returns the carrier and reads up to 80 characters from the keyboard and stores them in the I/O buffer in EBCDIC format. Upon recognition of the end-of-field character or reception of the 80th character, the routine returns control to the user (the remainder of the buffer is unchanged).

Upon recognition of the erase field character or the backspace character, the carriage is returned and the routine is re-initialized for the re-entry of the entire message. Characters are printed by the Console Printer during keyboard input.

Console Printer Output: The Subroutine returns the carrier and prints, from the I/O buffer, the number of characters indicated by Index Register 2.

Subroutines Loaded. The following subroutines are loaded along with TYPEZ:

HOLEZ, GETAD, EBCTB, HOLTB

WRTYZ – CONSOLE PRINTER OUTPUT SUBROUTINE

Buffer Size: Maximum of 120 words.

Operation. This subroutine returns the carrier and prints from the I/O buffer, the number of characters, indicated by Index Register 2.

Subroutines Loaded: The following subroutines are loaded along with WRTYZ:

GETAD, EBCTB

CARDZ – 1442 CARD READ PUNCH INPUT/OUTPUT SUBROUTINE

Buffer Size: Maximum of 80 words.

Card Input: This subroutine reads 80 columns from a card and stores the information in the I/O buffer in EBCDIC format.

Card Output: This subroutine punches, from the I/O buffer the number of characters indicated by Index Register 2. Punching is done in IBM card code format.

Subroutines Loaded: The following subroutines are loaded along with CARDZ:

HOLEX, GETAD, EBCTB, HOLTB

PAPTZ – 1134-1055 PAPER TAPE READER PUNCH I/O SUBROUTINE

Buffer Size: Maximum of 80 characters.

1134 Paper Tape Input: This subroutine reads paper tape punched in PTTC/8 format. The routine reads paper tape until 80 characters have been stored or until a new-line character is encountered. If 80 characters have been stored and a new-line character was not encountered, one more character, assumed to be a new line character, is read from tape. (Delete and case characters cause nothing to be stored.) If the first character read is not a case character, it is assumed to be a lower case character. The input is converted to EBCDIC Format.

1055 Paper Tape Output: The I/O buffer is converted from EBCDIC to PTTC/8, and the number of characters indicated by Index Register 2 is punched, in addition to the required case change characters.

PRNTZ – 1132 PRINTER OUTPUT SUBROUTINE

Buffer Size: Maximum of 121 characters.

Index Register 2: The value stored in Index Register 2 must be the number of characters to be printed, plus 1 because the first character in the I/O buffer is the carriage control character, followed by up to 120 characters to be printed. The first character to be printed is stored in location 003D.

The carriage of the 1132 Printer is controlled prior to the printing of a line. Following is a list of the carriage control characters and their related functions:

00F1	Skip to channel 1 prior to printing
00F0	Double space prior to printing
004E	No skip or space prior to printing

Any other character - Single space prior to printing.

Channel 12 Control: If a punch in channel 12 is encountered while a line is being printed, an automatic skip to channel 1 is taken prior to the printing of the next line.

DISKZ – DISK INPUT/OUTPUT SUBROUTINE

Operation: This subroutine reads or writes disk storage. Data is transferred to or from the disk, one sector (320 words) at a time.

Following a write operation, the subroutine performs a read back check on the data just written. If an error is detected, a re-write occurs. Similarly, if a sector is not located or an error is detected during a read, the subroutine repeats the operation. A read is attempted ten times before the computer halts with an error display.

Subroutines Loaded: No other subroutines are loaded along with DISKZ.

INTRODUCTION

The basic unit of information within the 1130 System is the 16-bit binary word. This information can be interpreted in a variety of ways, depending on the circumstances. For example, in internal computer operations, words may be interpreted as instructions, as addresses, as binary integers, or as floating-point numbers (see Arithmetic and Functional Subroutines).

A variety of data codes exists for the following reasons.

1. The programmer needs a compact notation to represent externally the bit configuration of each computer word. This representation is provided in the hexadecimal notation.
2. A code is required for representing alphameric (mixed alphabetic and numeric) data within the computer. This code is provided by the Extended Binary Coded Decimal Interchange Code (EBCDIC).
3. The design and operation of the input/output devices is such that many of them impose a unique correspondence between character representations in the external medium and the associated bit configurations within the computer. Subroutines are needed to convert input data from these devices to a form on which the computer can operate and to prepare computed results for output on the devices.

This and following sections of the manual describe the data codes used and the subroutines provided for converting data representations among these codes.

A detailed description of the binary, octal, hexadecimal, and decimal number systems is contained in the publication, IBM 1130 Functional Characteristics (Form A26-5881).

DESCRIPTIONS OF DATA CODES

In addition to the internal 16-bit binary representation, the conversion subroutines handle the following codes:

- Hexadecimal Notation
- IBM Card Code
- Perforated Tape and Transmission Code (PTTC/8)
- Console Printer Code
- Extended Binary Coded Decimal Interchange Code (EBCDIC)

A list of these codes is contained in Appendix D.

HEXADECIMAL NOTATION

Although binary numbers facilitate the operations of computers, they are bulky and awkward for the programmer to handle. A long string of 1's and 0's cannot be effectively transmitted from one individual to another. For this reason, the hexadecimal number system is often used as a shorthand method of communicating binary numbers. Because of the simple relationship of hexadecimal to binary, numbers can easily be converted from one system to another.

In hexadecimal notation a single digit is used to represent a 4-bit binary value as shown in Figure 5. Thus, a 16-bit word in the 1130 System can be expressed as four hexadecimal digits. For example, the binary value

1101001110111011

can be separated into four sections as follows:

Binary	1101	0011	1011	1011
Hexadecimal	D	3	B	B

Another advantage of hexadecimal notation is that fewer positions are required for output data printed, punched in cards, or punched in paper tape. In the example above, only four card columns are required to represent a 16-bit binary word.

BINARY	DECIMAL	HEXADECIMAL
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Figure 5. Hexadecimal Notation

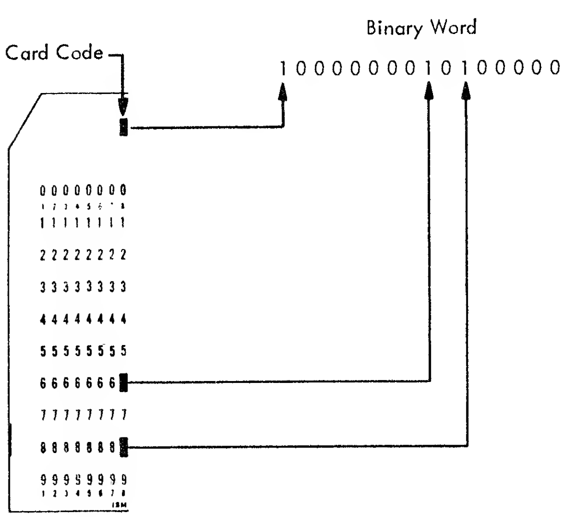
IBM CARD CODE

The IBM Card Code can be used as an input/output code with the 1442 Card Read-Punch and as an input code on the Input Keyboard.

This code defines a character by a combination of punches in a card column. Card-code data is taken from or placed into the leftmost twelve bits of a computer word as shown below:

Card Row	12	11	0	1	2	3	4	5	6	7	8	9	-	-	-	-
Computer Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

For example, a plus sign, which has a card code of 12, 6, 8, is placed into core storage in the binary configuration illustrated in the following diagram.



PERFORATED TAPE AND TRANSMISSION CODE (PTTC/8)

The PTTC/8 code is an 8-bit code used with IBM 1134/1055 Paper Tape units. This code represents a character by a stop position, a check position, and six positions representing the 6-bit code, BA8421. PTTC/8 characters can be packed two per computer word as shown below.

	1st								2nd							
PTTC/8 Characters	S	B	A	C	8	4	2	1	S	B	A	C	8	4	2	1
Computer Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The graphic character is defined by a combination of binary code and case; a control character is defined by a binary code and has the same meaning in upper or lower case. This implies that UC and LC characters must appear in a PTTC/8 message wherever necessary to establish or change the case.

The binary and PTTC/8 codes for 1/ (lower case) and =? (upper case) are shown in Figure 6.

The DEL and NL characters have a special meaning (in check mode only) when encountered by the paper tape subroutines.

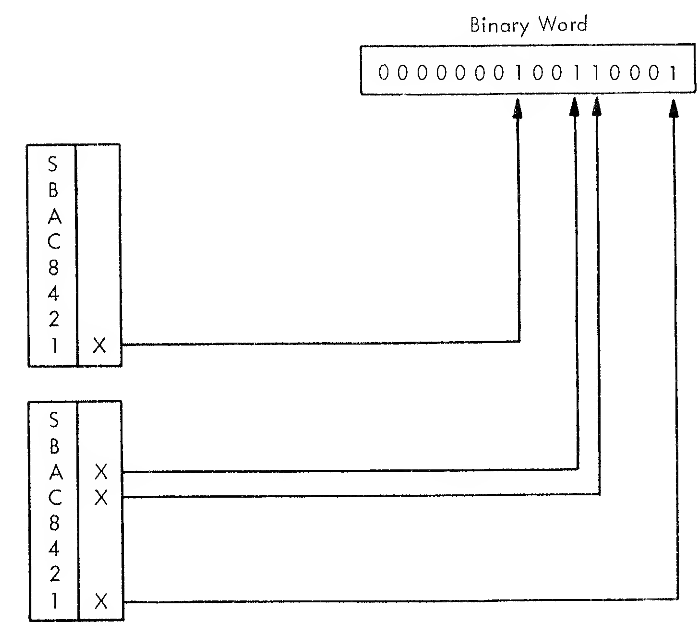


Figure 6. PTTC/8 Code for 1/ (Lower Case) or = ? (Upper Case)

CONSOLE PRINTER CODE

The console printer uses an 8-bit code that can be packed two per 16-bit word.

The following control characters have special meanings when used with the console printer.

Character	Control Operation
HT	Tabulate
RES	Shift to black ribbon
NL	Carrier return on new line
BS	Backspace
LF	Line feed without carrier return
RS	Shift to red ribbon

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC)

EBCDIC is the standard code for internal representation of alphameric and special characters and for the 1132 printer. The code occupies eight binary bits per character, making it possible to store either one or two characters per 16-bit word. The eight bits allow 256 possible codes. (At present, not all of these combinations represent characters.) The complete EBCDIC code is shown in Appendix D.

For reasons of efficiency, most of the conversion subroutines do not recognize all 256 codes. The asterisked codes in Appendix D constitute the subset recognized by most of the conversion subroutines.

CONVERSION SUBROUTINES

INTRODUCTION

Subroutines Included

The following data conversion subroutines are described in this section.

BINDC Binary value to IBM card code decimal value.

DCBIN IBM card code decimal value to binary value.

BINHX Binary value to IBM card code hexadecimal value.

HXBIN IBM card code hexadecimal value to binary value.

HOLEB IBM card code subset to EBCDIC subset; EBCDIC subset to IBM card code subset.

SPEED IBM card code characters to EBCDIC; EBCDIC to IBM card code characters.

PAPEB PTTC/8 subset to EBCDIC subset; EBCDIC subset to PTTC/8 subset.

PAPHL PTTC/8 subset to IBM card code subset; IBM card code subset to PTTC/8 subset.

PAPPR PTTC/8 subset to console printer code.

HOLPR IBM card code subset to console printer code.

EBPRT EBCDIC subset to console printer code.

The following conversion tables are used by some of the conversion subroutines.

PRTY Console printer code.

EBPA EBCDIC and PTTC/8 subsets.

HOLL IBM card code subset.

The first four subroutines change numeric data from its input form to a binary form, or from a binary form to an appropriate output data code. The last seven convert entire messages, one character at a time, from one input/output code to another. The types of conversions accomplished by these subroutines are illustrated in Figure 7.

Error Checking

All code conversion subroutines (except SPEED) accept only the codes marked with an asterisk in Appendix D. An input character that does not conform to a specified code is an error.

BINHX and BINDC subroutines do not detect errors. HXBIN and DCBIN terminate conversion at the point of error detection; they do not replace the character in error. The contents of the accumulator are meaningless when conversion is terminated because of an error.

CONVERTED FROM	CONVERTED TO								
	Binary	IBM Card Code (256)	IBM Card Code (Subset)	PTTC/8 (Subset)	EBCDIC (256)	EBCDIC (Subset)	Console Printer	Hex Equivalent (Card Code)	Decimal Equivalent (Card Code)
Binary								BINHX	BINDC
IBM Card Code (256)					SPEED				
IBM Card Code (Subset)				PAPHL		HOLEB	HOLPR		
PTTC/8 (Subset)			PAPHL			PAPEB	PAPPR		
EBCDIC (256)		SPEED							
EBCDIC (Subset)			HOLEB	PAPEB			EBPRT		
Hex Equivalent (Card Code)	HXBIN								
Decimal Equivalent (Card Code)	DCBIN								

Figure 7. Types of Conversions

The remainder of the conversion subroutines replace the character in error with a space character, stored in the output area in output code. Conversion is not terminated when an error is detected.

When a conversion subroutine detects an error it turns the Carry indicator off and turns the Overflow indicator on before returning control to the user. Otherwise, the settings of the Carry and Overflow indicators are not changed by the conversion subroutines.

BINDC

Description

This subroutine converts a 16-bit binary value to its decimal equivalent in five IBM card code numeric characters and one sign character. The five characters and the sign are placed in six computer words as illustrated in Figure 8.

Calling Sequence

```
            LIBF     BINDC
            DC       OUTPT
            .        .
OUTPT       BSS      6
```

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
A-Register	+01538	0000 0110 0000 0010
OUTPT	+	1000 0000 1010 0000
OUTPT + 1	0	0010 0000 0000 0000
OUTPT + 2	1	0001 0000 0000 0000
OUTPT + 3	5	0000 0001 0000 0000
OUTPT + 4	3	0000 0100 0000 0000
OUTPT + 5	8	0000 0000 0010 0000

Figure 8. BINDC Conversion

Input

Input is a 16-bit binary value in the accumulator.

Output

Output is an IBM card code sign character (plus or minus) in location OUTPT, and five IBM card code numeric characters in OUTPT +1 through OUTPT +5.

Errors Detected

The BINDC subroutine does not detect errors.

DCBIN

Description

This subroutine converts a decimal value in five IBM card code numeric characters and a sign character to a 16-bit binary word. The conversion is the reverse of the BINDC subroutine conversion illustrated in Figure 8.

Calling Sequence

```
          LIBF    DCBIN
          DC      INPUT
           .        .
           .        .
INPUT     BSS      6
```

Input

Input is an IBM card code sign character in location INPUT and five IBM card code decimal characters in INPUT +1 through INPUT +5.

Output

Output is a 16-bit binary word containing the converted value in the accumulator.

Errors Detected

Any sign other than an IBM card code plus, ampersand, space, or minus, or any decimal digits other than a space or 0 through 9 is an error. Any converted value greater than +32767 or less than -32768 is an error.

BINHX

Description

This subroutine converts a 16-bit binary word into hexadecimal notation in four IBM card code characters as illustrated in Figure 9.

I/O Locations	Conversion Data	Bits in Core Storage 0 ⟷ 15
Accumulator	A59E	1010 0101 1001 1110
OUTPT	A	1001 0000 0000 0000
OUTPT + 1	5	0000 0001 0000 0000
OUTPT + 2	9	0000 0000 0001 0000
OUTPT + 3	E	1000 0001 0000 0000

Figure 9. BINHX Conversion

Calling Sequence

```
          LIBF    BINHX
          DC      OUTPT
           .        .
           .        .
           .        .
OUTPT     BSS      4
```

Input

Input is a 16-bit binary word in the accumulator.

Output

Output is four IBM card code hexadecimal digits in location OUTPT through OUTPT +3.

Errors Detected

The BINHX subroutine does not detect errors.

HXBIN

Description

This subroutine converts four IBM card code hexadecimal characters into one 16-bit binary word. The conversion is the reverse of the BINHX subroutine conversion illustrated in Figure 9.

Calling Sequence

```
          LIBF    HXBIN
          DC      INPUT
           .        .
           .        .
           .        .
INPUT     BSS      4
```

Input

Input is four IBM card code hexadecimal digits in INPUT through INPUT +3.

Output

Output is a 16-bit binary word in the accumulator.

Errors Detected

Any input character other than an IBM card code 0 through 9 or A through F is an error.

HOLEB

Description

This subroutine converts IBM card code subset to the EBCDIC subset or converts the EBCDIC subset to IBM card code subset. Code conversion is illustrated in Figure 10.

Calling Sequence

```
          LIBF      HOLEB
          DC        /xxxx     (Control)
          DC        INPUT
          DC        OUTPT
          DC        nnnn      (Character count)
           .          .
           .          .
INPUT
OUTPT
```

Control Parameter

The control parameter consists of four hexadecimal digits. Digits 1-3 are not used. The fourth digit specifies the direction of conversion:

0 - IBM card code to EBCDIC
1 - EBCDIC to IBM card code

Input

Input is either IBM card code or EBCDIC characters, (as specified by the control parameter) starting in location INPUT. EBCDIC characters must be packed two characters per binary word. IBM card code characters are stored one character to each binary word.

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	JS	1101 0001 1110 0010
OUTPT	J	0101 0000 0000 0000
OUTPT + 1	S	0010 1000 0000 0000

Figure 10. HOLEB Conversion (EBCDIC to IBM Card Code)

Output

Output is either IBM card code or EBCDIC characters starting in location OUTPT. Characters are packed as described above.

If the direction of the conversion is IBM card code input to EBCDIC output, the input area can overlap the output area if the address INPUT is equal to or greater than the address OUTPT. If the direction of the conversion is EBCDIC input to IBM card code output, the input area can overlap the output area if the address INPUT + n/2 is equal to or greater than the address OUTPT + n, where n is the character count specified. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of characters to be converted; it is not equal to the number of binary words used for the EBCDIC characters because those characters are packed two per binary word. If an odd count is specified for EBCDIC output, bits 8 through 15 of the last word in the output area are not altered.

Errors Detected

Any input character not asterisked in Appendix D is an error.

SPEED

Description

This subroutine converts IBM card code to EBCDIC or EBCDIC to IBM card code. SPEED accepts all 256 characters defined in Appendix D.

If the input is IBM card code, the conversion time is much faster than that of HOLEB because a different conversion method is used when all 256

EBCDIC characters are accepted. If the SPEED subroutine is called before a card reading operation is completed, the SPEED subroutine synchronizes with a CARD subroutine read operation by checking bit 15 of the word to be processed before converting the word. If bit 15 is a one, the SPEED subroutine waits in a loop until the CARD subroutine sets the bit to a zero.

Calling Sequence

```
              LIBF          SPEED
              DC            /xxxx     (Control)
              DC            INPUT
              DC            OUTPT
              DC            nnnnn     (Character codes)
               .              .
               .              .
INPUT       |- - - - - - - - - - - -|
            |- - - - - - - - - - - -|
               .              .
               .              .
OUTPT       |- - - - - - - - - - - -|
            |- - - - - - - - - - - -|
```

Control Parameter

This parameter consists of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether the EBCDIC code is packed or unpacked.

0 – Packed, two EBCDIC characters per binary word

1 – Unpacked, one EBCDIC character per binary word (left-justified)

The fourth digit indicates the direction of conversion:

0 – IBM card code to EBCDIC

1 – EBCDIC to IBM card code

Input

Input is either IBM card code or EBCDIC characters (as specified by the control parameter) starting in location INPUT. EBCDIC characters can be packed or unpacked. IBM card code characters are stored one character to each binary word.

Output

Output is EBCDIC or IBM card code characters starting in location OUTPT. EBCDIC characters can be packed or unpacked; IBM card code characters are not packed.

The input area should not overlap the output area because of restart problems that can result from card feed errors.

Character Count

This parameter specifies the number of EBCDIC or IBM card code characters to be converted. If the character count is odd and the output code is EBCDIC, bits 8 through 15 of the last word are unaltered.

Errors Detected

Any input character code not listed in Appendix D is an error. All IBM card code punch combinations, except multiple punches in rows 1-7, are legal.

PAPEB

Description

This subroutine converts PTTC/8 subset to EBCDIC subset or EBCDIC subset to PTTC/8 subset. PAPEB conversion of EBCDIC to PTTC/8 with the initialize case option selected is illustrated in Figure 11.

Calling Sequence

```
              LIBF          PAPEB
              DC            /xxxx     (Control)
              DC            INPUT
              DC            OUTPT
              DC            nnnnn     (Character count)
               .              .
               .              .
INPUT       |- - - - - - - - - - - -|
            |- - - - - - - - - - - -|
               .              .
               .              .
OUTPT       |- - - - - - - - - - - -|
            |- - - - - - - - - - - -|
```

I/O Locations	Conversion Data	Bits in Core Storage 0 ⟵⟶ 15
INPUT	JS	1101 0001 1110 0010
OUTPT +0	UC J	0000 1110 0101 0001
+1	S DEL	0011 0010 0111 1111

Figure 11. PAPEB Conversion (EBCDIC to PTTC/8)

Control Parameter

This parameter consists of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether or not the case is to be initialized before conversion begins:

0 — Initialize case
1 — Do not alter case

The fourth digit indicates the direction of conversion:

0 — PTTC/8 to EBCDIC
1 — EBCDIC to PTTC/8

Input

Input (either PTTC/8 or EBCDIC characters, as specified by the control parameter) starts in location INPUT. Characters are packed two per 16-bit computer word in both codes.

Output

Output is either EBCDIC or PTTC/8 characters starting in OUTPT. Characters in either code are in packed format. The subroutine starts processing at location INPUT.

If the output is in EBCDIC, overlap of the input and output areas is possible if the address INPUT is equal to or greater than the address OUTPT.

If the output is in PTTC/8, overlap of the input and output areas is not recommended because the number of output characters might be greater than the number of input characters.

Character Count

This parameter specifies the number of PTTC/8 or EBCDIC characters in the input area. The count must include case shift characters even though they might not appear in the output. Because the input is packed, the character count will not be equal to the number of binary words in the input area. If an odd number of output characters is produced, bits 8-15 of the last used word in the output area are set to a space character if the output is EBCDIC, or to a delete character if the output is PTTC/8.

There is no danger of overflowing the output area if the number of words in a PTTC/8 output area is equal to the number of characters in the input area.

Errors Detected

Any input character that is not marked with an asterisk in Appendix D is an error.

Subroutine Operation

If the input is in PTTC/8 code, all control characters (except case shift (LC or UC) characters) are converted to output. Case shift characters only define the case mode of the graphic characters that follow.

If the initialize option is selected, the case is set to lower. All characters are interpreted as lower case characters until an upper case shift (UC) character is encountered. If the do-not-alter option is selected, the case remains set according to the last case shift character encountered in the previous LIBF message.

If the input is in EBCDIC, all data and control characters are converted to output. The user should not specify case shifting in his input message; this is handled automatically by the PAPEB subroutine.

Case shift characters are inserted in a PTTC/8 output message where needed to define certain graphic characters that have the same binary value and are differentiated only by a case mode character. For example, the binary value 0101 1011 (5B), is interpreted as a $ in lower case and an ! in upper case (see Appendix D).

If the initialize option is selected, the case shift character needed to interpret the first graphic character is inserted in the output message and the case mode is initialized for that mode. If the do-not-alter option is selected, the case mode remains set according to the last case shift character required in the previous LIBF message, i.e., no case shift is forced.

If a case shift character appears in the input message, it is output but does not affect the case mode. If it is an upper case shift (UC) and the next input character requires an upper case shift, the subroutine still inserts an upper case shift into the message, i.e., two UC characters will appear in the output message.

The conversion is halted whenever the character count is decremented to zero or whenever a new line (NL) control character is detected.

PAPHL

Description

This subroutine converts PTTC/8 subset to IBM card code subset or IBM card code subset to PTTC/8

subset. Figure 12 illustrates the relationship of the two codes for converting PTTC/8 to IBM card code.

Calling Sequence

```
              LIBF      PAPHL
              DC        /xxxx    (Control)
              DC        INPUT
              DC        OUTPT
              DC        nnnnn    (Character count)
               .          .
               .          .
               .          .
INPUT       |- - - - - - - - - - -|
            |- - - - - - - - - - -|
               .          .
               .          .
OUTPT       |- - - - - - - - - - -|
            |- - - - - - - - - - -|
```

Control Parameter

This parameter consists of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether or not the case is to be initialized before conversion begins:

0 – Initialize case
1 – Do not alter case

The fourth digit indicates the type of conversion:

0 – PTTC/8 to IBM card code
1 – IBM card code to PTTC/8

Input

Input is either PTTC/8 or IBM card code characters (as specified by the control parameter) starting in location INPUT. PTTC/8 characters are packed two per binary word; IBM card code characters are not packed.

Output

Output is either IBM card code or PTTC/8 code characters starting in location OUTPT. PTTC/8 codes are packed two per binary word; IBM card code characters are not packed.

If the conversion is IBM card code input to PTTC/8 output, the input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. Case shift characters are inserted in the output message where needed to define certain graphic characters (see PAPEB Subroutine).

If the conversion is PTTC/8 input to IBM card code output, the input area may overlap the output area if the address INPUT + (n/2) is equal to or greater than the address OUTPT + n, where n is the character count. The subroutine starts processing at location INPUT.

Character Count

This parameter specifies the number of PTTC/8 or EBCDIC characters in the input area. The count must include case shift characters, even though they might not appear in the output. Because the input may be packed, the character count may not be equal to the number of binary words in the input area.

There is no danger of overflowing the output area confines if the number of words in the output area is equal to the number of characters in the input area.

Errors Detected

Any input character not marked by an asterisk in Appendix D is an error.

Subroutine Operation

Case and shift character handling is described under PAPEB.

If an odd number of PTTC/8 output characters is produced, bits 8-15 of the last used word in the output area are set to a delete character.

The conversion is halted whenever the character count is decremented to zero or whenever a new line (NL) control character is detected.

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	UC J	0000 1110 0101 0001
	S T	0011 0010 0010 0011
OUTPT	J	0101 0000 0000 0000
OUTPT +1	S	0010 1000 0000 0000
OUTPT +2	T	0010 0100 0000 0000

Figure 12. PAPHL Conversion (PTTC/8 to IBM Card Code)

PAPPR

Description

This subroutine converts PTTC/8 subset to console printer code. The conversion is illustrated in Figure 13.

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	UC J	0000 1110 0101 0001
INPUT +1	LC $	0110 1110 0101 1011
OUTPT	J $	0111 1100 0100 0000

Figure 13. PAPPR Conversion

Calling Sequence

```
         LIBF     PAPPR
         DC       /xxx0    (Control)
         DC       INPUT
         DC       OUTPT
         DC       nnnnn    (Character count)
           .        .
INPUT    |- - - - - - - - - - -|
         |- - - - - - - - - - -|
           :        :
OUTPT    |- - - - - - - - - - -|
         |- - - - - - - - - - -|
```

Control Parameter

This parameter consists of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether or not the case is to be initialized before conversion begins:

0 — Initialize case
1 — Do not alter case

The fourth digit must be zero, specifying console printer code.

Input

Input consists of PTTC/8 characters starting in location INPUT. PTTC/8 characters are packed two per binary word. All control characters except case shift (LC or UC) characters are converted to output. Case shift characters are used only to define the case mode of the graphic characters that follow.

Output

Output consists of console printer characters starting in location OUTPT. This code is packed two characters per binary word. If overlap of the input and output areas is desired, the address INPUT must be equal to or greater than the address OUTPT. This is necessary because the subroutine starts processing at location INPUT.

Character Count

This parameter specifies the number of PTTC/8 characters in the input area. The count must include case shift characters, even though they do not appear in the output. Because the input is packed, the character count is not equal to the number of binary words in the input area.

If an odd number of output characters is produced, bits 8-15 of the last used word in the output area are set to a space character.

The conversion is halted whenever the character count is decremented to zero or whenever a new line (NL) control character is detected.

Errors Detected

Any input character not marked by an asterisk in Appendix D is an error.

HOLPR

Description

This subroutine converts IBM card code subset to console printer code. The conversion is illustrated in Figure 14.

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	J	0101 0000 0000 0000
INPUT + 1	?	0010 0000 0110 0000
OUTPT	J ?	0111 1100 1000 0110

Figure 14. HOLPR Conversion

Calling Sequence

```
        LIBF        HOLPR
        DC          /xxx0      (Control)
        DC          INPUT
        DC          OUTPT
        DC          nnnnn      (Character count)
         .            .
         .            .
INPUT
         .            .
         .            .
OUTPT
```

Control Parameter

This parameter consists of four hexadecimal digits. Digits 1-3 are not used. The fourth digit must be a zero, specifying console printer code.

Input

Input consists of IBM card code characters, starting in location INPUT. The characters are not packed.

Output

Output consists of console printer code characters, starting in location OUTPT. The code is packed two characters per binary word.

The input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of IBM card code characters to be converted and is equal to the number of words in the input area. If an odd count is specified, bits 8-15 of the last word used in the output area are not altered.

Errors Detected

Any input character not marked with an asterisk in Appendix D is an error.

EBPRT

Description

This subroutine converts EBCDIC subset to console printer characters. The conversion is shown in Figure 15.

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	L E	1101 0011 1100 0101
INPUT + 1	E S	1100 0101 1110 0010
OUTPT	L E	0101 1100 0011 0100
OUTPT + 1	E S	0011 0100 1001 1000

Figure 15. EBPRT Conversion

Calling Sequence

```
        LIBF        EBPRT
        DC          /xxx0      (Control)
        DC          INPUT
        DC          OUTPT
        DC          nnnnn      (Character count)
         .            .
         .            .
INPUT
         .            .
         .            .
OUTPT
```

Control Parameter

This parameter consists of four hexadecimal digits. Digits 1-3 are not used. The fourth digit must be a zero, specifying console printer code.

Input

Input consists of EBCDIC characters starting in location INPUT. EBCDIC characters are packed two per word.

Output

Output consists of console printer characters starting in location OUTPT. The code is packed two characters per binary word.

The address INPUT must be equal to or greater than the address OUTPT if overlap of the input and output areas is desired. The subroutine starts processing at location INPUT.

Character Count

This parameter specifies the number of EBCDIC characters to be converted. This count is not equal to the number of words in the input area. If an odd count is specified, bits 8-15 of the last word used in the output area are not altered.

Errors Detected

Any input character not marked with an asterisk in Appendix D is an error.

The IBM 1130 Subroutine Library includes the arithmetic and functional subroutines that are the most frequently required because of their general applicability. There are 44 subroutines, some of which have several entry points.

Table 4 lists the arithmetic and functional subroutines that are included in the Subroutine Library.

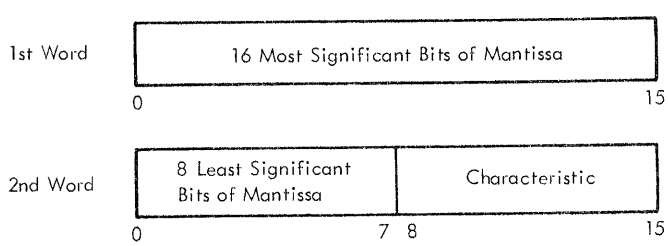
FLOATING-POINT DATA FORMATS

Many of the IBM 1130 arithmetic and functional subroutines offer two ranges of precision: standard and extended. The standard precision provides 23 significant bits, and the extended precision provides up to 31 significant bits.

To achieve correct results from a particular subroutine, the input arguments must be in the proper format.

Standard Precision Format

Standard precision floating-point numbers are stored in core storage as shown below:

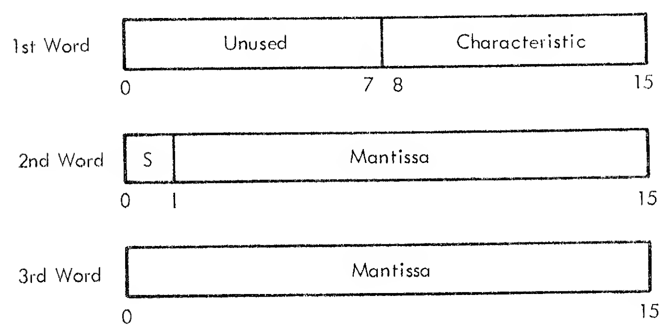


Numbers can consist of up to 23 significant bits (mantissa) with a binary exponent ranging from -128 to +127. Two adjacent storage locations are required for each number. The first (lowest) location must be even-numbered. The sign of the mantissa is in bit zero of the first word. The next 23 bits represent the mantissa (2's complement) and the remaining 8 bits represent the characteristic. The mantissa is normalized to fractional form, i. e. , the implied binary point is between bits zero and one.

The characteristic is formed by adding +128 to the exponent. For example, an exponent of -32 is represented by a characteristic of 128-32, or 96. An exponent of +100 is represented by a characteristic of 100 + 128, or 228. Since $128_{10} = 80_{16}$ the characteristic of a nonnegative exponent always has a 1-bit in position 1, while the characteristic of a negative exponent always produces a 0-bit in position 1. A normal zero consists of all zero bits in both the characteristic and the mantissa.

Extended Precision Format

Extended precision floating-point numbers are stored in three adjacent core locations as shown below:



Numbers can consist of up to 31 significant bits with a binary exponent ranging from -128 to +127.

Bits zero through seven of the first word are unused; bits eight through 15 of the first word represent the characteristic of the exponent (formed in the same manner as in the standard range format); bit zero of the second word contains the sign of the mantissa; and the remaining 31 bits represent the mantissa (2 ' s complement).

Fixed Point Format

Fractional numbers, as applied to the fixed-point subroutines, XSQR, XMDS, XMD, and XDD, are defined as binary fractions with implied binary points of zero. That is, the binary point is positioned between the sign (bit 0) and the most significant bit (bit 1).

The user can consider the binary point to be in any position in his fixed-point numbers. To correctly interpret the results the following rules must be observed.

1. Only numbers with binary points in equivalent positions can be correctly added or subtracted.
2. The binary point location in the product of two numbers is the sum of the binary point locations of the multiplier and the multiplicand.
3. The binary point location in the quotient of two numbers is the difference between the binary point locations of the dividend and the divisor.
4. The binary point location in a number that is input to the fixed-point square root subroutine (XSQR) must be an even number from 0-14. The binary point location in the output root is half the binary point location of the input number.

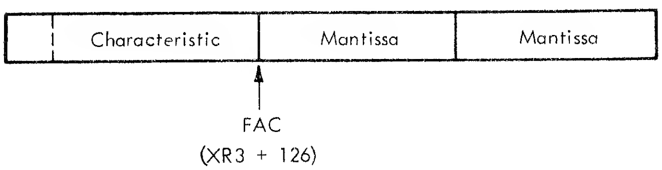
Table 4. Arithmetic and Functional Subroutines

SUBROUTINE	NAME	
Floating-Point	Standard Precision	Extended Precision
Add/Subtract	*FADD/*FSUB	*EADD/*ESUB
Multiply	*FMPY	*EMPY
Divide	*FDIV	*EDIV
Load/Store FAC	*FLD/*FSTO	*ELD/*ESTO
Trigonometric Sine/Cosine	FSINE/FCOSN, FSIN/FCOS	ESINE/ECOSN, ESIN/ECOS
Trigonometric Arctangent	FATN, FATAN	EATN, EATAN
Square Root	FSQR, FSQRT	ESQR, ESQRT
Natural Logarithm	FLN, FALOG	ELN, EALOG
Exponential (e^x)	FXPN, FEXP	EXPN, EEXP
Hyperbolic Tangent	FTNH/FTANH	ETNH/ETANH
Floating-Point Base to an Integer Exponent	*FAXI	*EAXI
Floating-Point Base to a Floating-Point Exponent	*FAXB	*EAXB
Floating-Point to Integer	IFIX	IFIX
Integer to Floating-Point	FLOAT	FLOAT
Normalize	NORM	NORM
Floating Binary to Decimal/Floating Decimal to Binary	FBTD/FDTB	FBTD/FDTB
Floating-Point Arithmetic Range Check	FARC	FARC
Fixed-Point		
Integer Base to an Integer Exponent	*FIXI	*FIXI
Fixed-Point Square Root	XSQR	XSQR
Fixed-Point Fractional Multiply (short)	XMDS	
Fixed-Point Double Word Multiply	XMD	XMD
Fixed-Point Double Word Divide	XDD	XDD
Special Function		
Floating-Point Reverse Subtract	*FSBR	*ESBR
Floating-Point Reverse Divide	*FDVR	*EDVR
Floating-Point Reverse Sign	SNR	SNR
Floating-Point Absolute Value	FAVL, FABS	EAVL, EABS
Integer Absolute Value	IABS	IABS
Miscellaneous		
Get Parameters	FGETP	EGETP

NOTE: By adding an X to those names prefixed with an asterisk, the user can cause the contents of Index Register 1 to be added to the address of the argument specified in the subroutine calling sequence to form the effective argument address. For example, FADDX would be the modified form of FADD.

FLOATING-POINT PSEUDO ACCUMULATOR

IBM 1130 floating-point subroutines sometimes require a register or accumulator that can accommodate numbers in floating-point format. Since all of the 1130 registers are only 16 bits in length, a pseudo accumulator must be set up to contain two- or three-word floating-point numbers. The pseudo accumulator (designated FAC for floating accumulator) is a three-word register occupying the three highest locations of the transfer vector (see IBM 1130 Assembler Language, C26-5927). The user can refer to these words by using Index Register 3 plus a fixed displacement (XR3 + 125, 126, or 127). The format of the FAC is shown below.



The effective address of the mantissa is always even.

NOTE: Arithmetic and functional subroutines do not save and restore the contents of the 1130 accumulator or the accumulator extension. The main program should provide for this if the contents are significant.

CALLING SEQUENCES

The arithmetic and functional subroutines are called via a CALL or LIBF statement followed, in some cases, by a DC statement containing the actual or symbolic address of an argument. In the descriptions that follow, the notations (ARG) and (FAC) refer to the contents of the operand rather than its address. The name FAC refers to the floating-point pseudo accumulator. The extended precision subroutine names are prefixed with the letter E (subroutines which handle both precisions have the same name and do not have a prefix).

Note also that some of the functional subroutines can be called via two different calling sequences. One calling sequence assumes the argument is in FAC; the other specifies the location of the argument with a DC statement.

In addition, some subroutines can have indexed linkage to the argument. The calling sequence is the same except for the subroutine name which contains an X suffix. Also, some subroutines perform more than one type of arithmetic or function. For example, FSIN and FCOS are different entry points to the same subroutine. Each subroutine is listed in Table 4 with the corresponding entry points.

Floating-Point Add

LIBF	FADD, FADDX, EADD or EADDX
DC	ARG
Input	Floating-point augend in FAC Floating-point addend in location ARG
Result	(FAC) + (ARG) replaces (FAC)

Floating-Point Subtract

LIBF	FSUB, FSUBX, ESUB or ESUBX
DC	ARG
Input	Floating-point minuend in FAC Floating-point subtrahend in location ARG
Result	(FAC) - (ARG) replaces (FAC)

Floating-Point Multiply

LIBF	FMPY or EMPY
DC	ARG
Input	Floating-point multiplicand in FAC Floating-point multiplier in location ARG
Result	(FAC) times (ARG) replaces (FAC)

Floating-Point Divide

LIBF	FDIV, FDIVX, EDIV or EDIVX
DC	ARG
Input	Floating-point dividend in FAC Floating-point divisor in location ARG
Result	(FAC) / (ARG) replaces (FAC)

Load FAC

LIBF	FLD, FLDX, ELD or ELDX
DC	ARG
Input	Floating-point number in location ARG
Result	(ARG) replaces (FAC)

Store FAC

LIBF	FSTO, FSTOX, ESTO or ESTOX
DC	ARG
Input	Floating-point number in FAC
Result	(FAC) replaces (ARG)

Floating-Point Trigonometric Sine

CALL	FSINE or ESINE
Input	Floating-point argument (in radians) in FAC
Result	Sine of (FAC) replaces (FAC)

or

CALL	FSIN or ESIN
DC	ARG
Input	Floating-point argument (in radians) in location ARG
Result	Sine of (ARG) replaces (FAC)

Floating-Point Trigonometric Cosine

CALL	FCOSN or ECOSN
Input	Floating-point argument (in radians) in FAC
Result	Cosine of (FAC) replaces (FAC)

or

CALL	FCOS or ECOS
DC	ARG
Input	Floating-point argument (in radians) in location ARG
Result	Cosine of (ARG) replaces (FAC)

Floating-Point Trigonometric Arctangent

CALL	FATN or EATN
Input	Floating-point argument in FAC
Result	Arctangent of (FAC) replaces (FAC); the result lies within the range $\pm\frac{\pi}{2}$ radians (±90 degrees)

or

CALL	FATAN or EATAN
DC	ARG
Input	Floating-point argument in location ARG
Result	Arctangent of (ARG) replaces (FAC); the result lies within the range $\pm\frac{\pi}{2}$ radians (±90 degrees)

Floating-Point Square Root

CALL	FSQR or ESQR
Input	Floating-point argument in FAC
Result	Square root of (FAC) replaces (FAC)

or

CALL	FSQRT or ESQRT
DC	ARG
Input	Floating-point argument in location ARG
Result	Square root of (ARG) replaces (FAC)

Floating-Point Natural Logarithm

CALL	FLN or ELN
Input	Floating-point argument in FAC
Result	Log_e (FAC) replaces (FAC)

or

CALL	FALOG or EALOG
DC	ARG
Input	Floating-point argument in location ARG
Result	Log_e (ARG) replaces (FAC)

Floating-Point Exponential

CALL	FXPN or EXPN
Input	Floating-point argument in FAC = n
Result	e^n replaces (FAC)

or

CALL	FEXP or EEXP
DC	ARG
Input	Floating-point argument in location ARG = n
Result	e^n replaces (FAC)

Floating-Point Hyperbolic Tangent

CALL	FTNH or ETNH
Input	Floating-point argument in FAC
Result	TANH (FAC) replaces (FAC)

or

CALL	FTANH or ETANH
DC	ARG
Input	Floating-point argument in location ARG
Result	TANH (ARG) replacees (FAC)

Floating-Point Base to an Integer Exponent

LIBF	FAXI, FAXIX, EAXI, or EAXIX
DC	ARG
Input	Floating-point base in FAC Integer exponent in location ARG
Result	(FAC), raised to the exponent contained in ARG, replaces (FAC)

Floating-Point Base to a Floating-Point Exponent

CALL	FAXB, FAXBX, EAXB or EAXBX
DC	ARG
Input	Floating-point base in FAC Floating-point exponent in location ARG
Result	(FAC) raised to the exponent contained in ARG replaces (FAC)

Floating-Point to Integer

LIBF	IFIX
Input	Floating-point number in FAC
Result	Integer in the Accumulator

Integer to Floating-Point

LIBF	FLOAT
Input	Integer in the Accumulator
Result	Floating-point number in FAC

Normalize

LIBF	NORM
Input	Floating-point unnormalized number in FAC
Result	The mantissa portion of FAC is shifted until the most significant bit resides in bit position 1. The characteristic is changed to reflect the number of bit positions shifted.

Floating Binary to Decimal

CALL	FBTD
DC	LDEC
Input	Floating-point number in FAC
Output	A string of EBCDIC-coded decimal data, starting at location LDEC. Each EBCDIC character occupies the rightmost eight bits of a word. The output format is exactly as follows. sd. ddddddddEsdd where s represents a sign (plus or minus) and d represents one of the decimal digits 0-9.

Floating Decimal to Binary

CALL	FDTB
DC	LDEC
Input	Same as output from FBTD subroutine. The input field may not contain any embedded blanks. The first blank encountered is interpreted as the end of the string.
Output	Floating-point number in FAC

Floating-Point Arithmetic Range Check

LIBF	FARC
Result	This subroutine checks for floating-point overflow or underflow, and sets programmed indicators for interrogation by a FORTRAN program.

Integer Base to an Integer Exponent

LIBF	FIXI or FIXIX
DC	ARG
Input	Integer base in the accumulator Integer exponent in location ARG
Result	(Accumulator) raised to the exponent contained in ARG replaces (accumulator)

Fixed-Point Square Root

CALL	XSQR
Input	Fixed-point fractional argument (16 bits only) in the accumulator.
Result	Square root of (accumulator) replaces (accumulator). If the argument is negative the absolute value is used and the overflow indicator is turned on.

Fixed-Point Double-Word Multiply

LIBF	XMD
Input	Double-word fractional multiplier in FAC (addressed by XR3 + 126) Double-word fractional multiplicand in the accumulator and extension
Result	Double-word fractional product in the accumulator and extension A and Q

Fixed-Point Fractional Multiply (Short)

LIBF	XMDS
Input	Double-word fractional multiplier in the accumulator and extension Double-word fractional multiplicand in FAC (addressed by XR3 + 126)
Result	Product in the accumulator and extension (XMDS is shorter and faster than XMD; however, the resulting precision is 24 bits).

Fixed-Point Double-Word Divide

LIBF	XDD
Input	Double-word fractional dividend in FAC (addressed by XR3 + 126) Double-word fractional divisor in accumulator and extension
Result	Double-word fractional quotient in the accumulator and extension. The double dividend in FAC is destroyed by the execution of the subroutine.

Floating-Point Reverse Subtract

LIBF	FSBR, FSBRX, ESBR or ESBRX
DC	ARG
Input	Floating minuend in location ARG Floating subtrahend in FAC
Result	(ARG) - (FAC) replaces (FAC)

Floating-Point Reverse Divide

LIBF	FDVR, FDVRX, EDVR or EDVRX
DC	ARG
Input	Floating dividend in location ARG Floating divisor in FAC
Result	(ARG) / (FAC) replaces (FAC)

Floating-Point Reverse Sign

LIBF	SNR
Input	Floating point number, X, in FAC
Result	-X replaces X in FAC

Floating-Point Absolute Value

CALL	FAVL or EAVL
Input	Floating point number, X, in FAC
Result	Absolute value of X replaces X in FAC

or

CALL	FABS or EABS
DC	ARG
Input	Floating point number, X, in location ARG
Result	Absolute value of X replaces (FAC)

Integer Absolute Value

CALL	IABS
Input	An integer, I, in the accumulator
Result	Absolute value of I replaces I in the accumulator

Get Parameters (FGETP or EGETP)

Example:

```
MAIN    CALL    SUBR
        DC      ARG
NEXT    etc.
  .       .       .
  .       .       .
  .       .       .
SUBR    DC      0
        LIBF    FGETP or EGETP
SUBEX   DC      0
        etc.
  .       .       .
  .       .       .
  .       .       .
  .       .       .
        BSC I   SUBEX
```

The FGETP subroutine performs two functions for a subroutine accessed by a CALL statement. It loads FAC with the contents of ARG; it sets SUBEX to return to NEXT in the main program.

ARITHMETIC AND FUNCTIONAL SUBROUTINE ERROR INDICATORS

The highest three-word entry in the transfer vector is reserved for the floating-point pseudo accumulator (FAC). The next to highest three-word entry is reserved for the arithmetic and functional subroutine error indicators.

The first word (addressed XR3 + 122) of the second entry is used for floating-point arithmetic overflow and underflow indicators. The second word (XR3 + 123) is used for a divide check indicator, and the third word (XR3 + 124) is used for functional subroutine indicators. The loader initializes all three words to zero.

Word One

Each floating point subroutine checks for exponent underflow and overflow. If either occurs, word one and FAC are set as follows.

1, if overflow has occurred (FAC = ± maximum).
3, if underflow has occurred (FAC = zero).

The last error condition replaces any previous indication. Also, when an underflow occurs, FAC is set to zero.

When an overflow occurs, FAC is set to the largest valid number of the same algebraic sign as the contents of FAC when the overflow was detected.

Word Two

The floating-point divide subroutines check for division by zero. If this occurs, word two is set to 1. The dividend is not changed.

Word Three

The functional subroutines check for the following error conditions and set word three as described.

Floating-Point Square Root

When the argument is negative, the square root of the argument's absolute value is returned, and a bit is ORed into position 13 of word three.

Floating-Point Natural Logarithm

When the argument is zero, FAC is set to the largest negative value and a bit is ORed into position 15 of word three. When the argument is negative, the absolute value of the argument is used and a bit is ORed into position 15 of word three.

Floating-Point Trigonometric Sine and Cosine

When the absolute value of the argument is equal to or greater than 2^{24}, FAC is set to zero and a bit is ORed into position 14 of word three.

Floating-Point to Integer

When the absolute value of the argument is greater than $2^{15}-1$, the largest possible signed result is placed in the accumulator and a bit is ORed into position 12 of word three.

Integer Base to an Integer Exponent

When the base is zero and the exponent is zero or negative, a zero result is returned and a bit is ORed into position 11 of word three.

Floating-Point Base to an Integer Exponent

When the base is zero and the exponent is zero or negative, a zero result is returned and a bit is ORed into position 10 of word three.

Floating-Point Base Raised to a Floating-Point Exponent

When the base is zero and the exponent is zero or negative, a zero result is returned and a bit is ORed into position 9 of word three. When the base is negative and the exponent is not zero, the absolute value of the base is used and a bit is ORed into position 15 of word three.

FUNCTIONAL SUBROUTINE ACCURACY

Given:

$e \equiv$ Maximum error
$f(x) \equiv$ True value of the function
$f^*(x) \equiv$ Value generated by subroutine
$(<+\infty) \equiv\ \leq$ Largest valid floating-point number
$(>-\infty) \equiv\ \geq$ Most negative floating-point number

EXTENDED PRECISION SUBROUTINES

The following statements of accuracy apply to extended precision subroutines.

ESIN

$$e \equiv \left| \frac{\sin(x) - \sin^*(x)}{x} \right| < 3.0 \times 10^{-9}$$

for the range

$$-1.0 \times 10^{6} \leq x < 0$$

$$1.0 \times 10^{6} \geq x > 0$$

for $x = 0$ $\sin(x) \equiv 0$

ECOS

$$e \equiv \left| \frac{\cos(x) - \cos^*(x)}{|x| + \frac{\pi}{2}} \right| < 3.0 \times 10^{-9}$$

for the range

$$-1.0 \times 10^{6} \leq x \leq 1.0 \times 10^{6}$$

EATAN

$$e \equiv \left| \frac{\operatorname{atn}(x) - \operatorname{atn}^*(x)}{\operatorname{atn}(x)} \right| < 2.0 \times 10^{-9}$$

for the range

$$-3.88336148 \times 10^{37} \leq x \leq 3.88336148 \times 10^{37}$$

EEXP

$$e \equiv \left| \frac{e^{x} - (e^{x})^*}{e^{x}} \right| < \left\{ \begin{matrix} 2.0 \times 10^{-9} \; |x| \\ \text{or} \\ 2.0 \times 10^{-9} \end{matrix} \right\} \begin{matrix} \text{whichever} \\ \text{is} \\ \text{greater} \end{matrix}$$

for the range

$$-\ln(\infty) < x < \ln(\infty)$$

i.e., $0 < e^{x} < \infty$

ELN

$$e \equiv \left| \frac{\ln(x) - \ln^*(x)}{\ln(x)} \right| < 3.0 \times 10^{-9}$$

for the range

$$0 < x < \infty$$

ETANH

$$e \equiv \left| \tanh(x) - \tanh^*(x) \right| < 3.0 \times 10^{-9}$$

for the range

$$-\infty < x < \infty$$

ESQRT

$$e \equiv \left| \frac{\sqrt{x} - \sqrt{x}^*}{\sqrt{x}} \right| < 1.0 \times 10^{-9}$$

for the range

$$0 < x < \infty$$

STANDARD PRECISION SUBROUTINES

The following statements of accuracy apply to the standard precision subroutines.

FSIN

$$e \equiv \left| \frac{\sin(x) - \sin^*(x)}{x} \right| < 2.5 \times 10^{-7}$$

for the range

$$-1.0 \times 10^{6} \leq x < 0$$

$$1.0 \times 10^{6} \geq x > 0$$

for $x = 0$ $\sin(x) \equiv 0$

FCOS

$$e \equiv \left| \frac{\cos(x) - \cos^*(x)}{|x| + \frac{\pi}{2}} \right| < 2.5 \times 10^{-7}$$

for the range

$$-1.0 \times 10^{6} \leq x \leq 1.0 \times 10^{6}$$

FATAN

$$e \equiv \left| \frac{\operatorname{atn}(x) - \operatorname{atn}^*(x)}{\operatorname{atn}(x)} \right| < 5.0 \times 10^{-7}$$

for the range

$$-3.883361 \times 10^{37} \leq x \leq 3.883361 \times 10^{37}$$

FEXP

$$e \equiv \left| \frac{e^x - (e^x)^*}{e^x} \right| < \left\{ \begin{array}{c} 2.5 \times 10^{-7} \, |x| \\ \text{or} \\ 2.5 \times 10^{-7} \end{array} \right\} \begin{array}{c} \text{whichever} \\ \text{is} \\ \text{greater} \end{array}$$

for the range

$$-\ln(\infty) < x < \ln(\infty) \text{ i.e., } 0 < e^x < \infty$$

FLN

$$e \equiv \left| \frac{\ln(x) - \ln^*(x)}{\ln(x)} \right| < 4.0 \times 10^{-7}$$

for the range

$$0 < x < \infty$$

FTANH

$$e \equiv \left| \tanh(x) - \tanh^*(x) \right| < 2.5 \times 10^{-7}$$

for the range

$$-\infty < x < +\infty$$

FSQRT

$$e \equiv \left| \frac{\sqrt{x} - \sqrt{x}^*}{\sqrt{x}} \right| < 2.5 \times 10^{-7}$$

for the range

$$0 < x < \infty$$

ELEMENTARY FUNCTION ALGORITHMS

The choice of an approximating algorithm for a given function depends on such considerations as expected execution time, storage requirements, and accuracy. For a given accuracy, and within reasonable limits, storage requirements vary inversely as the execution time. Polynomial approximating is used to evaluate the elementary functions to effect the desired balance between storage requirements and efficiency.

SINE-COSINE

Polynomial Approximation

Given a floating point number, x, n and y are defined such that

$$x\left(\frac{\pi}{2}\right) = n + y$$

where n is an integer and $0 \leq y < 1$. Thus, $x = 2\pi n + 2\pi y$, and the identities are

$$\sin x = \sin 2\pi y \text{ and } \cos x = 2\pi y .$$

The polynomial approximation, $F(z)$, for the function $(\sin 2\pi z)/z$ is used where $-1/4 \leq z \leq 1/4$.

The properties of sines and cosines are used to compute these functions as follows.

$$\cos 2\pi y = F(z)$$

where:

$z = 1/4 - y$ in the range $0 \le y \le 1/2$
$z = y - 3/4$ in the range $1/2 \le y < 1$

$$\sin 2\pi y = F(z)$$

where:

$z = y$ in the range $0 \le y < 1/4$
$z = 1/2 - y$ in the range $1/4 \le y < 3/4$
$z = y - 1$ in the range $3/4 \le y < 1$

Extended Precision

$$F(z) = z(a_1 + a_2 z^2 + a_3 z^4 + a_4 z^6 + a_5 z^8 + a_6 z^{10}$$

where

$a_1 = 6.2831853071$
$a_2 = -41.341702117$
$a_3 = 81.605226206$
$a_4 = -76.704281321$
$a_5 = 42.009805726$
$a_6 = -14.394135365$

Standard Precision

$$F(z) = a_1 z + a_2 z^3 + a_3 z^5 + a_4 z^7 + a_5 z^9$$

where:

$a_1 = 6.2831853$
$a_2 = -41.341681$
$a_3 = 81.602481$
$a_4 = -76.581285$
$a_5 = 39.760722$

ARCTANGENT

Polynomial Approximation

The routine for arctangent is built around a polynomial, $F(z)$, that approximates Arctan(z) in the range $-.23 \le z \le .23$. The Arctan(z) for z outside this range is found by using the identities:

$$\text{Arctan}(-z) = -\text{Arctan}(z)$$

$$\text{Arctan}(z) = a_k + \text{Arctan}\left[\frac{z - b_k}{zb_k + 1}\right]$$

where

$$a_k = \frac{k\pi}{7} \quad \text{and} \quad b_k = \tan a_k$$

and k is determined so that

$$\tan\frac{(2k-1)\pi}{14} \le z < \tan\frac{(2k+1)\pi}{14} \qquad k = 1, 2, 3$$

Having determined the value of k appropriate to z, the transformation $x = z - b_k / zb_k + 1$ puts x in the range $-\tan \pi/14 \le x < \tan \pi/14$. The polynomial $F(z)$ was chosen to be good over a range slightly larger (i.e., .23   tan $\pi/14$) so that the comparisons to determine the interval in which z lies need be only standard precision accuracy.

Extended Precision

$$F(z) = x\,(1.0 - a_1 x^2 + a_2 x^4 - a_3 x^6 + a_4 x^8)$$

$a_1 = .33333327142$
$a_2 = .19999056792$
$a_3 = .14235177463$
$a_4 = .09992331248$

Standard Precision

$$F(z) = x\,(1.0 - .333329573z^2$$

$$+ .199641035z^4 - .131779888z^6)$$

SQUARE ROOT

Square Root (x)
Let $x = 2^{2b}F$ when $.25 \leq F < 1$
then $\sqrt{X} = 2^b \sqrt{F}$
where $\sqrt{F} = P_i$ i = number of approximation

$P_1 = AF + B$ as a first approximation followed by 2 Newton iterations

where

$A = .875,\ B = .27863$ when $.25 \leq F < .5$

or

$A = .578125,\ B = .421875$ when $.5 \leq F < 1$

$$P_2 = \frac{\left(P_1 + \frac{F}{P_1}\right)}{2}$$

$$P_3 = \frac{\left(P_2 + \frac{F}{P_2}\right)}{2}$$

NATURAL LOGARITHM

Polynomial Approximation

Given a normalized floating point number

$$x = 2^k \times f\left(\frac{1}{2} \leq f < 1\right),$$

j and g are found such that $x = 2^j g$ where $(\sqrt{2}/2 \leq g < \sqrt{2})$. This is done by setting $j = k-1$, $g = 2f$ if $f < \sqrt{2}/2$ and $j = k$, $g = f$ otherwise.

Thus:

$$\ln(x) = j \cdot \ln(2) + \ln(g).$$

The approximation for $\ln(g)$, $\sqrt{2}/2 \leq g < \sqrt{2}$, is based on the series

$$\ln\frac{v+x}{v-x} = 2\left[(x/v) + (x^3/3v^3) + (x^5/5v^5) + \ldots\right]$$

which converges for $(-v < x < v)$.
With the transformation

$$x = v\frac{f-1}{f+1}, \quad v = (\sqrt{2} + 1)^2$$

so that $-1 \leq x < 1$ for $\sqrt{2}/2 \leq g < \sqrt{2}$.
Substituting,

$$\ln(g) = 2\,(z + z^3/3 + z^5/5 + \ldots)$$

where $z = x/v = (f-1)/(f+1)$. The approximation used is $G(z)$ for $\ln(g)/z$ in the range $\sqrt{2}/2 \leq g < \sqrt{2}$.

Extended Precision

$$G(z) = b_0 + b_2z^2 + b_4z^4 + b_6z^6 + b_8z^8$$

$b_0 = 2.0$
$b_2 = .666666564181$
$b_4 = .400018840613$
$b_6 = .28453572660$
$b_8 = .125$
$z = \frac{g-1}{g+1}$
$\sqrt{2}/2 = .7071067811865$
$\ln(2) = .6931471805599$

Thus, the required calculation is:

$$\ln(x) = j \cdot \ln(2) + zG(z)$$

Standard Precision

$$G(z) = 2.0 + .66664413786\,z^2$$
$$+ .4019234697z^4 + .25z^6$$

The IBM 1130 Subroutine Library includes three dump subroutines: Dump Selected Data on the console printer, Dump Selected Data on the 1132 Printer, and Dump Status Area. These subroutines allow the user to dump selected portions of core storage during the execution of an object program.

DUMP SELECTED DATA ON CONSOLE PRINTER OR 1132 PRINTER

Two subroutines are available to select an area of core storage and dump it out on the console printer or the 1132 Printer. Each of these subroutines has two entry points, one for hexadecimal output and one for decimal output. The entry points for the various configurations are shown below:

DMTX0 Dump on console printer in hexadecimal form, using the WRTY0 subroutine

DMTD0 Dump on console printer in decimal form, using the WRTY0 subroutine

DMPX1 Dump on 1132 Printer in hexadecimal form, using the PRNT1 subroutine

DMPD1 Dump on 1132 Printer in decimal form, using the PRNT1 subroutine

Calling Sequence

The calling sequence for any of the above functions is as follows:

```
CALL     ENTRY POINT
DC       START
DC       END
```

START and END represent the starting and ending addresses of the portion of core storage to be dumped. A starting address greater than the ending address results in the error message, ERROR IN ADDRESS, and a return to the main program.

Format

Before the actual dump appears on the selected output device, the user is given one line of status information. This line indicates the status of the Overflow and Carry triggers (ON or OFF), the contents of the Accumulator and Extension, and the contents of the three index registers. The index register contents are given in both hexadecimal and decimal form, regardless of which type of output was requested. The format of the status information is shown below:

OFF	ON	HHHH (±DDDDD)	HHHH (±DDDDD)
Overflow	Carry	Accumulator	Extension

HHHH (±DDDDD)	HHHH (±DDDDD)	HHHH (±DDDDD)
Index Register 1	Index Register 2	Index Register 3

All other data is dumped eight words to a line; the address of the first word in each line is printed to the left of the line. Hexadecimal data is printed four characters per word; decimal data is printed five digits per word, preceded by a plus or minus sign.

Page numbers are not printed for either subroutine. However, the 1132 Printer subroutine does provide for automatic page overflow upon the sensing of a channel 12 punch in the carriage tape.

DUMP STATUS AREA

This subroutine provides a relatively easy and efficient means of dumping the first 80 words of core storage. These words contain status information relating to index registers, interrupt addresses, interval timers, etc., which may be required frequently during the testing of a program. It may also be desirable to dump these words before loading because pressing the Load key destroys the data in the first 80 words of core storage.

The Dump Status Area subroutine is called via the following statement:

```
CALL     DMP80
```

The console printer prints the first 80 words of core storage in hexadecimal form; the printing format provides spacing between words. After typing the last word, the subroutine returns control to the main program.

INTERRUPT SERVICE SUBROUTINES

The following rules must be adhered to when writing an ISS:

1. Precede the ISS statement with an LIBR statement if the subroutine is to be called by LIBF rather than CALL.
2. Precede the subroutine with an EPR (extended) or an SPR (standard) statement if precision specification is necessary.
3. Precede the subroutine with one ISS statement defining the entry point (one only), the ISS number, and the ILS subroutines required. The device interrupt level assignments, and the ISS numbers used in the IBM-provided ISS and ILS routines, are shown in Table 5.
4. The entry points of an ISS are defined by the related ILS. This must be taken into consideration when a user-written ISS is used with an IBM supplied ILS. The ILS executes a Branch and Store I instruction to the ISS at the ISS entry point plus n (see Table 5). The ISS must return to the ILS via a BSC instruction (not a BOSC).

Table 5. ISS/ILS Correspondence

ISS Number	Device	Device Interrupt Level Assignments	n
1	1442 Card Reader Punch	0, 4	+4, +7
2	Input Keyboard/ Console Printer	4	+4
3	1134/1055 Paper Tape Reader/Punch	4	+4
4	Disk Storage	2	+4
6	1132 Printer	1	+4
7	1627 Plotter	3	+4

INTERRUPT LEVEL SUBROUTINES

The following rules must be adhered to when writing an ILS:

1. Precede the subroutine with an ILS statement identifying the interrupt level involved.
2. Precede all instructions by an ISS branch table and include one word per ILSW bit used. If the ILSW will not be scanned, (i. e., a single ISS routine to handle all interrupts on the level), then a one word table is sufficient. The minimum table size is one word. Table words must be non-zero.

$$\left.\begin{array}{l}\text{ILSW Bit 15 word}\\ \text{ILSW Bit 14 word}\\ \quad\vdots\\ \text{ILSW Bit 0 word}\end{array}\right\}\ \text{ISS Branch Table}$$

The ISS branch table identifies both the ISS subroutine and the point within the ISS which should be entered for each bit used in the ILSW. The actual linkage is generated by the relocating loader or core image converter. Basic to this generation is the ISS number implied by bits 8-15 of the branch table word and specified in the ISS statement. This number identifies a core location in which the loader or converter has stored the address of the called entry point in the ISS. This entry point address is incremented by the value in bits 0-7 of the branch table word, producing the branch linkage. The loader or converter replaces the ISS branch table word with the generated branch linkage.

At execution time the ISS branch table contains actual addresses. It may be used with an indirect branch and store I (BSI) instruction to reach the ISS corresponding to that ILSW bit position. The ILSW bit that is ON can be determined by the execution of a SLCA instruction. At the completion of this instruction, the index register specified contains a relative value

equivalent to the bit position in the ISS branch table. An indirect, indexed BSI may then be used to reach the appropriate ILS.

Each word in the ISS branch table has the following format:

Bits 0-7 — Increment added to the entry point named in the ISS statement to obtain the interrupt entry point in the ISS for this ILSW bit. (In IBM-written ISS subroutines, this increment is +4 for the primary interrupt level and +7 for the second interrupt level.)

Bits 8-15 — Address of the loader interrupt transfer vector for the ISS subroutine for this ILSW bit (equal to ISS number +51). This address should match word 13 of the compressed ISS header card.

3. The ILS entry point must immediately follow the ISS branch address table and must be loaded as a zero. The loader assumes that the first zero word in the program is the end of the branch table and is also the entry point of the ILS. (The table must contain at least one entry.) The interrupt results in a BSI to the ILS entry point.

4. To clear the level, a user-written ILS, used with an IBM-supplied ISS, should exit via the return linkage with a BOSC instruction.

OVERLAY ROUTINES (FLIPPERS)

The monitor subroutine library contains two flipper routines which are used to call LOCAL (load on call) routines into core storage. FLIP0 is used with DISK0 and DISKZ, and FLIP1 is used with DISK1 and DISKN. FLIP0 reads a LOCAL into storage one sector at a time, whereas FLIP1 passes the total word count to DISK1 or DISKN and that routine reads in the entire LOCAL. When a LOCAL routine is called, control is passed to the flipper routine which reads the LOCAL into core storage if it is not already in core and transfers control to it. All LOCALs in a given core load are executed from the same core storage locations; each LOCAL overlays the previous one.

EXPONENTIAL

Polynomial Approximation

To find e^x, the following identity is used.

$$e^x = 2^{\left(x \log_2 e\right)}$$

To reduce the range, we let

$$x \log_2 e = n + d + z$$

where:

- n is the integral portion of the real number,
- d is a discreet fraction (1/8, 3/8, 5/8, or 7/8) of the real number, and
- z is the remainder which is in the range $-1/8 \leq z \leq 1/8$.

Thus,

$$e^x = 2^n \times 2^d \times 2^z$$

and it is necessary to only approximate 2^z for $-1/8 \leq z \leq 1/8$ by using the polynomial $F(z)$.

Extended Precision

$$F(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4 + a_5 z^5$$

where:

$a_0 = 1.0$

$a_1 = .69314718057$

$a_2 = .24022648580$

$a_3 = .055504105406$

$a_4 = .0096217398747$

$a_5 = .0013337729375$

Standard Precision

$$F(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4$$

where:

$a_0 = 1.0$

$a_1 = .693147079$

$a_2 = .240226486$

$a_3 = .0555301557$

$a_4 = .00962173985$

HYPERBOLIC TANGENT

$$\text{Tanh}(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

for

$x \geq 32$	$\text{Tanh}(x) = 1$
$x \leq -32$	$\text{Tanh}(x) = -1$

FLOATING-POINT BASE TO AN INTEGER EXPONENT

$$A = e^{\ln A}$$

therefore:

$$A^B = \left(e^{\ln A}\right)^B = e^{B \ln A}$$

Subroutine	Names
FORTRAN	
Called by CALL	
Loader Reinitialization (card only)	LOAD
Data Switch	DATSW
Sense Light On	SLITE, SLITT
Overflow Test	OVERF
Divide Check Test	DVCHK
Function Test	FCTST
Trace Start	TSTRT
Trace Stop	TSTOP
Integer Transfer of Sign	ISIGN
Real Transfer of Sign (E)	ESIGN
Real Transfer of Sign (S)	FSIGN
Called by LIBF (card/paper tape)	
Real IF Trace (E)	VIF
Real IF Trace (S)	WIF
Integer IF Trace (E)	VIIF
Integer IF Trace (S)	WIIF
Integer Arithmetic Trace (E)	VIAR, VIARX
Integer Arithmetic Trace (S)	WIAR, WIARX
Real Arithmetic Trace (E)	VARI, VARIX
Real Arithmetic Trace (S)	WARI, WARIX
Computed GOTO Trace (E)	VGOTO
Computed GOTO Trace (S)	WGOTO
Trace Test-Set Indicator	TTEST, TSET
Pause	PAUSE
Stop	STOP
Subscript Calculation	SUBSC
Store Argument Address	SUBIN
I/O Linkage (E)	VFIO, VRED, VWRT, VCOMP, VIOAI, VIOAF, VIOFX, VIOIX, VIOF, VIOI
I/O Linkage (S)	WFIO, WRED, WWRT, WCOMP, WIOAI, WIOAF, WIOFX, WIOIX, WIOF, WIOI
Card Input/Output	CARDZ
Printer-Keyboard Output	WRTYZ
Printer-Keyboard Input/Output	TYPEZ
1132 Printer Output	PRNTZ
Paper Tape Input/Output	PAPTZ
Card Code-EBCDIC Conversion	HOLEZ
Console Printer Code Table	EBCTB
Card-Keyboard Code Table	HOLTB
Address Calculation	GETAD
Called by LIBF (monitor)	
Real IF Trace (E)	SEIF
Real IF Trace (S)	SFIF
Integer IF Trace	SIIF
Integer Arithmetic Trace	SIAR, SIARX
Real Arithmetic Trace (E)	SEAR, SEARX
Real Arithmetic Trace (S)	SFAR, SFARX
Computed GOTO Trace	SGOTO
Trace Test-Set Indicator	TTEST, TSET
Pause	PAUSE
Stop	STOP
Subscript Calculation	SUBSC
Store Argument Address	SUBIN
I/O Linkage (non-disk)	SFIO, SRED, SWRT, SCOMP, SIOAF, SIOAI, SIOF, SIOI, SIOFX, SIOIX
Disk-I/O Linkage	SDFIO, SDRED, SDWRT, SDCOM, SDAF, SDAI, SDF, SDI, SDFX, SDIX
Disk Find	SDFND
Card Input/Output	CARDZ
Printer-Keyboard Output	WRTYZ
Printer-Keyboard Input/Output	TYPEZ
1132 Printer Output	PRNTZ
Paper Tape Input/Output	PAPTZ
Card Code-EBCDIC Conversion	HOLEZ
Console Printer Code Table	EBCTB
Card-Keyboard Code Table	HOLTB
Address Calculation	GETAD
ARITHMETIC AND FUNCTIONAL	
Called by CALL	
Floating-Point Hyperbolic Tangent (E)	ETNH, ETANH
Floating-Point Hyperbolic Tangent (S)	FTNH, FTANH
Floating-Point Base to Floating-Point Exponent (E)	EAXB, EAXBX
Floating-Point Base to Floating-Point Exponent (S)	FAXB, FAXBX
Floating-Point Natural Logarithm (E)	ELN, EALOG
Floating-Point Natural Logarithm (S)	FLN, FALOG
Floating-Point Exponential (E)	EXPN, EEXP
Floating-Point Exponential (S)	FXPN, FEXP
Floating-Point Square Root (E)	ESQR, ESQRT
Floating-Point Square Root (S)	FSQR, FSQRT
Floating-Point Trigonometric Sine/Cosine (E)	ESIN, ESINE, ECOS, ECOSN
Floating-Point Trigonometric Sine/Cosine (S)	FSIN, FSINE, FCOS, FCOSN
Floating-Point Trigonometric Arctangent (E)	EATN, EATAN
Floating-Point Trigonometric Arctangent (S)	FATN, FATAN
Fixed-Point Square Root	XSQR
Floating-Point Absolute Value (E)	EAVL, EABS
Floating-Point Absolute Value (S)	FAVL, FABS
Integer Absolute Value	IABS
Floating Binary to Decimal/Floating Decimal to Binary	FBTD, FDTB
Called by LIBF	
Get Parameters (E)	EGETP
Get Parameters (S)	FGETP
Floating-Point Base to Integer Exponent (E)	EAXI, EAXIX
Floating-Point Base to Integer Exponent (S)	FAXI, FAXIX
Floating-Point Reverse Divide (E)	EDVR, EDVRX
Floating-Point Reverse Divide (S)	FDVR, FDVRX
Floating-Point Divide (E)	EDIV, EDIVX
Floating-Point Divide (S)	FDIV, FDIVX
Floating-Point Multiply (E)	EMPY, EMPYX
Floating-Point Multiply (S)	FMPY, FMPYX
Floating-Point Reverse Subtract (E)	ESBR, ESBRX
Floating-Point Reverse Subtract (S)	FSBR, FSBRX
Floating-Point Add/Subtract (E)	EADD, EADDX, ESUB, ESUBX
Floating-Point Add/Subtract (S)	FADD, FADDX, FSUB, FSUBX
Load/Store FAC (E)	ELD, ELDX, ESTO, ESTOX
Load/Store FAC (S)	FLD, FLDX, FSTO, FSTOX
Fixed Point Double Word Divide	XDD
Fixed Point Double Word Multiply	XMD
Fixed Point Fractional Multiply (short)	XMDS
Floating-Point Reverse Sign	SNR
Integer to Floating-Point	FLOAT
Floating-Point to Integer	IFIX
Fixed Integer Base to an Integer Exponent	FIXI, FIXIX
Normalize	NORM
Floating-Point Arithmetic Range Check	FARC
DUMP	
Called by CALL	
Dump Status Area	DMP80
Selective Dump on Console Printer	DMTX0, DMTD0
Selective Dump on Printer	DMPX1, DMPD1
DISK SUBROUTINE INITIALIZE (card/paper tape only)	
Called by CALL	
Set Pack Initialize Routine	SPIR0, SPIR1, SPIRN
OVERLAY (monitor only)	
Called by LIBF	
Local Read-in	FLIP0, FLIP1
INTERRUPT SERVICE	
Called by LIBF	
Card	CARD0, CARD1
Disk (part of supervisor in monitor system)	DISK0, DISK1, DISKN
Paper Tape	PAPT1, PAPTN
Plotter	PLOT1
1132 Printer	PRNT1
Console Printer-Keyboard	TYPE0, WRTY0
INTERRUPT LEVEL (card/paper tape only)	
Level 0	ILS00*
Level 1	ILS01*
Level 2	ILS02*
Level 3	ILS03*
Level 4	ILS04*
*These subroutines are not identified by name in the card and paper tape systems	
CONVERSION	
Called by LIBF	
Binary to Decimal	BINDC
Binary to Hexadecimal	BINHX
Decimal to Binary	DCBIN
EBCDIC to Console Printer Code	EBPRT
IBM Card Code to or From EBCDIC	HOLEB
IBM Card Code to Console Printer Code	HOLPR
Hexadecimal to Binary	HXBIN
EBCDIC to or from PTTC/8	PAPEB
IBM Card Code to or from PTTC/8	PAPHL
PTTC/8 to Console Printer Code	PAPPR
IBM Card Code to or from EBCDIC	SPEED
EBCDIC and PTTC/8 Table	EBPA
IBM Card Code Table	HOLL
Console Printer Code Table	PRTY

APPENDIX B. ERRORS DETECTED BY THE ISS SUBROUTINES

ERROR	CONTENTS OF ACCUMULATOR: Binary	CONTENTS OF ACCUMULATOR: Hexadecimal	Contents of Extension (if any)
Card			
*Last card	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0	
*Feed check *Read check *Punch check	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 1	
Device not ready Last card indicator on for Read	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0	
Illegal device (not 0 version) Device not in system Illegal function Word count over +80 Word count zero or negative	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1	1 0 0 1	
Printer-Keyboard			
Device not ready	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2 0 0 0	
Device not in system Illegal function Word count zero or negative	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1	2 0 0 1	
Paper Tape			
*Punch not ready	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 4	
*Reader not ready	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	0 0 0 5	
Device not ready	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	3 0 0 0	
Illegal device Illegal function Word count zero or negative Illegal check digit	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1	3 0 0 1	
Disk			
*Disk overflow	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 4	
*Seek failure remaining after ten attempts	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 3	Effective Sector Id
*Read check remaining after ten attempts Data Error Data overrun	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 1	Effective Sector Id
*Write check remaining after ten attempts Write select Data error Data Overrun	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 2	Effective Sector Id
Device not ready	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0	5 0 0 0	
Illegal device (not 0 version) Device not in system Illegal function Attempt to write in file protected area Word count zero or negative Word count over +320 (0 version only) Starting sector identification over + 1599	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1	5 0 0 1	
1132 Printer			
*Channel 9 detected	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 3	
*Channel 12 detected	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 4	
Device not ready or end of forms	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	6 0 0 0	
Illegal function Illegal word count	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1	6 0 0 1	
Plotter			
Plotter not ready	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	7 0 0 0	
Illegal device Device not in system Illegal function Word count zero or negative	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1	7 0 0 1	

NOTE: The errors marked with an asterisk cause a branch via the error parameter. These errors are detected during the processing of interrupts; as a consequence, the user error routine is an interrupt routine, executed at the priority level of the I/O device.

All other errors cause a branch to location 41. The address of the LIBF in error is in location 40.

APPENDIX C. SUBROUTINE ACTION AFTER RETURN FROM A USER'S ERROR ROUTINE

Error Code	Condition	Subroutine Action
Card		
0000	If function is PUNCH	Eject card and terminate
	Otherwise	Terminate Immediately
0001*	If Accumulator is 0	Terminate immediately
	Otherwise	Loop until 1442 is ready, then reinitiate operation
Paper Tape		
0004, 0005	If Accumulator is 0	Terminate immediately
	Otherwise	Check again for device ready
Disk		
0001, 0002, and 0003	If A Reg. is 0	Terminate immediately
	Otherwise	Retry 10 more times
1132 Printer		
0003, and 0004	If Accumulator is 0	Terminate immediately
	Otherwise	Skip to channel 1 and then terminate

*Assumes operator intervention.

APPENDIX D. CHARACTER CODE CHART

Ref No.	EBCDIC Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	IBM Card Code Hex	Graphics and Control Names		1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex Notes
0	0000	0000	00	12		0	9	8	1	B030	NUL				
1		0001	01	12			9		1	9010					
2		0010	02	12			9		2	8810					
3		0011	03	12			9		3	8410					
4		0100	04	12			9		4	8210	PF	Punch Off			
5*		0101	05	12			9		5	8110	HT	Horiz. Tab		6D ⑤	41 ①
6*		0110	06	12			9		6	8090	LC	Lower Case		6E ⑤	
7*		0111	07	12			9		7	8050	DEL	Delete		7F ⑤	
8		1000	08	12			9	8		8030					
9		1001	09	12			9	8	1	9030					
10		1010	0A	12			9	8	2	8830					
11		1011	0B	12			9	8	3	8430					
12		1100	0C	12			9	8	4	8230					
13		1101	0D	12			9	8	5	8130					
14		1110	0E	12			9	8	6	80B0					
15		1111	0F	12			9	8	7	8070					
16	0001	0000	10	12	11		9	8	1	D030					
17		0001	11		11		9		1	5010					
18		0010	12		11		9		2	4810					
19		0011	13		11		9		3	4410					
20*		0100	14		11		9		4	4210	RES	Restore		4C ⑤	05 ②
21*		0101	15		11		9		5	4110	NL	New Line		DD ⑤	81 ③
22*		0110	16		11		9		6	4090	BS	Backspace		5E ⑤	11
23		0111	17		11		9		7	4050	IDL	Idle			
24		1000	18		11		9	8		4030					
25		1001	19		11		9	8	1	5030					
26		1010	1A		11		9	8	2	4830					
27		1011	1B		11		9	8	3	4430					
28		1100	1C		11		9	8	4	4230					
29		1101	1D		11		9	8	5	4130					
30		1110	1E		11		9	8	6	40B0					
31		1111	1F		11		9	8	7	4070					
32	0010	0000	20		11	0	9	8	1	7030					
33		0001	21			0	9		1	3010					
34		0010	22			0	9		2	2810					
35		0011	23			0	9		3	2410					
36		0100	24			0	9		4	2210	BYP	Bypass			
37*		0101	25			0	9		5	2110	LF	Line Feed		3D ⑤	03
38*		0110	26			0	9		6	2090	EOB	End of Block		3E ⑤	
39		0111	27			0	9		7	2050	PRE	Prefix			
40		1000	28			0	9	8		2030					
41		1001	29			0	9	8	1	3030					
42		1010	2A			0	9	8	2	2830					
43		1011	2B			0	9	8	3	2430					
44		1100	2C			0	9	8	4	2230					
45		1101	2D			0	9	8	5	2130					
46		1110	2E			0	9	8	6	20B0					
47		1111	2F			0	9	8	7	2070					
48	0011	0000	30	12	11	0	9	8	1	F030					
49		0001	31				9		1	1010					
50		0010	32				9		2	0810					
51		0011	33				9		3	0410					
52		0100	34				9		4	0210	PN	Punch On			
53*		0101	35				9		5	0110	RS	Reader Stop		0D ⑤	09 ④
54*		0110	36				9		6	0090	UC	Upper Case		0E ⑤	
55		0111	37				9		7	0050	EOT	End of Trans.			
56		1000	38				9	8		0030					
57		1001	39				9	8	1	1030					
58		1010	3A				9	8	2	0830					
59		1011	3B				9	8	3	0430					
60		1100	3C				9	8	4	0230					
61		1101	3D				9	8	5	0130					
62		1110	3E				9	8	6	00B0					
63		1111	3F				9	8	7	0070					

NOTES: Typewriter Output
① Tabulate
② Shift to black
③ Carrier Return
④ Shift to red
⑤ The Same in Either Upper or Lower Case

* Recognized by all Conversion subroutines

Codes that are not asterisked are recognized only by the SPEED subroutine

Ref No.	EBCDIC			IBM Card Code							Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex
	Binary		Hex	Rows						Hex				
	0123	4567		12	11	0	9	8	7-1					
64*	0100	0000	40	no punches						0000	(spoce)	‡	10 5	21
65		0001	41	12		0	9		1	B010				
66		0010	42	12		0	9		2	A810				
67		0011	43	12		0	9		3	A410				
68		0100	44	12		0	9		4	A210				
69		0101	45	12		0	9		5	A110				
70		0110	46	12		0	9		6	A090				
71		0111	47	12		0	9		7	A050				
72		1000	48	12		0	9	8		A030				
73		1001	49	12				8	1	9020				
74*		1010	4A	12				8	2	8820	¢		20 (U)	02
75*		1011	4B	12				8	3	8420	. (period)	4B	6B (L)	00
76*		1100	4C	12				8	4	8220	<		02 (U)	DE
77*		1101	4D	12				8	5	8120	(	4D	19 (U)	FE
78*		1110	4E	12				8	6	80A0	+	4E	70 (U)	DA
79*		1111	4F	12				8	7	8060	ǀ (logicol OR)		3B (U)	C6
80*	0101	0000	50	12						8000	&	50	70 (L)	44
81		0001	51	12	11		9		1	D010				
82		0010	52	12	11		9		2	C810				
83		0011	53	12	11		9		3	C410				
84		0100	54	12	11		9		4	C210				
85		0101	55	12	11		9		5	C110				
86		0110	56	12	11		9		6	C090				
87		0111	57	12	11		9		7	C050				
88		1000	58	12	11		9	8		C030				
89		1001	59		11			8	1	5020				
90*		1010	5A		11			8	2	4820	!		5B (U)	42
91*		1011	5B		11			8	3	4420	$	5B	5B (L)	40
92*		1100	5C		11			8	4	4220	*	5C	08 (U)	D6
93*		1101	5D		11			8	5	4120	)	5D	1A (U)	F6
94*		1110	5E		11			8	6	40A0	;		13 (U)	D2
95*		1111	5F		11			8	7	4060	¬ (logical NOT)		6B (U)	F2
96*	0110	0000	60		11					4000	- (dash)	60	40 (L)	84
97*		0001	61			0			1	3000	/	61	31 (L)	BC
98		0010	62		11	0	9		2	6810				
99		0011	63		11	0	9		3	6410				
100		0100	64		11	0	9		4	6210				
101		0101	65		11	0	9		5	6110				
102		0110	66		11	0	9		6	6090				
103		0111	67		11	0	9		7	6050				
104		1000	68		11	0	9	8		6030				
105		1001	69			0		8	1	3020				
106		1010	6A	12	11					C000				
107*		1011	6B			0		8	3	2420	, (commo)	6B	3B (L)	80
108*		1100	6C			0		8	4	2220	%		15 (U)	06
109*		1101	6D			0		8	5	2120	_ (underscore)		40 (U)	BE
110*		1110	6E			0		8	6	20A0	>		07 (U)	46
111*		1111	6F			0		8	7	2060	?		31 (U)	86
112	0111	0000	70	12	11	0				E000				
113		0001	71	12	11	0	9		1	F010				
114		0010	72	12	11	0	9		2	E810				
115		0011	73	12	11	0	9		3	E410				
116		0100	74	12	11	0	9		4	E210				
117		0101	75	12	11	0	9		5	E110				
118		0110	76	12	11	0	9		6	E090				
119		0111	77	12	11	0	9		7	E050				
120		1000	78	12	11	0	9	8		E030				
121		1001	79					8	1	1020				
122*		1010	7A					8	2	0820	:		04 (U)	82
123*		1011	7B					8	3	0420	#		0B (L)	C0
124*		1100	7C					8	4	0220	@		20 (L)	04
125*		1101	7D					8	5	0120	' (opostrophe)	7D	16 (U)	E6
126*		1110	7E					8	6	00A0	=	7E	01 (U)	C2
127*		1111	7F					8	7	0060	"		0B (U)	E2

‡ Any code other than those defined will be interpreted by PRNT1 as a space.

Ref No.	EBCDIC			IBM Card Code							Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex
	Binary		Hex	Rows						Hex				
	0123	4567		12	11	0	9	8	7-1					
128	1000	0000	80	12		0		8	1	B020				
129		0001	81	12		0			1	B000	a			
130		0010	82	12		0			2	A800	b			
131		0011	83	12		0			3	A400	c			
132		0100	84	12		0			4	A200	d			
133		0101	85	12		0			5	A100	e			
134		0110	86	12		0			6	A080	f			
135		0111	87	12		0			7	A040	g			
136		1000	88	12		0		8		A020	h			
137		1001	89	12		0	9			A010	i			
138		1010	8A	12		0		8	2	A820				
139		1011	8B	12		0		8	3	A420				
140		1100	8C	12		0		8	4	A220				
141		1101	8D	12		0		8	5	A120				
142		1110	8E	12		0		8	6	A0A0				
143		1111	8F	12		0		8	7	A060				
144	1001	0000	90	12	11			8	1	D020				
145		0001	91	12	11				1	D000	j			
146		0010	92	12	11				2	C800	k			
147		0011	93	12	11				3	C400	l			
148		0100	94	12	11				4	C200	m			
149		0101	95	12	11				5	C100	n			
150		0110	96	12	11				6	C080	o			
151		0111	97	12	11				7	C040	p			
152		1000	98	12	11			8		C020	q			
153		1001	99	12	11		9			C010	r			
154		1010	9A	12	11			8	2	C820				
155		1011	9B	12	11			8	3	C420				
156		1100	9C	12	11			8	4	C220				
157		1101	9D	12	11			8	5	C120				
158		1110	9E	12	11			8	6	C0A0				
159		1111	9F	12	11			8	7	C060				
160	1010	0000	A0		11	0		8	1	7020				
161		0001	A1		11	0			1	7000				
162		0010	A2		11	0			2	6800	s			
163		0011	A3		11	0			3	6400	t			
164		0100	A4		11	0			4	6200	u			
165		0101	A5		11	0			5	6100	v			
166		0110	A6		11	0			6	6080	w			
167		0111	A7		11	0			7	6040	x			
168		1000	A8		11	0		8		6020	y			
169		1001	A9		11	0	9			6010	z			
170		1010	AA		11	0		8	2	6820				
171		1011	AB		11	0		8	3	6420				
172		1100	AC		11	0		8	4	6220				
173		1101	AD		11	0		8	5	6120				
174		1110	AE		11	0		8	6	60A0				
175		1111	AF		11	0		8	7	6060				
176	1011	0000	B0	12	11	0		8	1	F020				
177		0001	B1	12	11	0			1	F000				
178		0010	B2	12	11	0			2	E800				
179		0011	B3	12	11	0			3	E400				
180		0100	B4	12	11	0			4	E200				
181		0101	B5	12	11	0			5	E100				
182		0110	B6	12	11	0			6	E080				
183		0111	B7	12	11	0			7	E040				
184		1000	B8	12	11	0		8		E020				
185		1001	B9	12	11	0	9			E010				
186		1010	BA	12	11	0		8	2	E820				
187		1011	BB	12	11	0		8	3	E420				
188		1100	BC	12	11	0		8	4	E220				
189		1101	BD	12	11	0		8	5	E120				
190		1110	BE	12	11	0		8	6	E0A0				
191		1111	BF	12	11	0		8	7	E060				

Ref No.	EBCDIC Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	IBM Card Code Hex	Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex
192	1100	0000	C0	12		0				A000	(+ zero)			
193*		0001	C1	12					1	9000	A	C1	61 (U)	3C or 3E
194*		0010	C2	12					2	8800	B	C2	62 (U)	18 or 1A
195*		0011	C3	12					3	8400	C	C3	73 (U)	1C or 1E
196*		0100	C4	12					4	8200	D	C4	64 (U)	30 or 32
197*		0101	C5	12					5	8100	E	C5	75 (U)	34 or 36
198*		0110	C6	12					6	8080	F	C6	76 (U)	10 or 12
199*		0111	C7	12					7	8040	G	C7	67 (U)	14 or 16
200*		1000	C8	12				8		8020	H	C8	68 (U)	24 or 26
201*		1001	C9	12			9			8010	I	C9	79 (U)	20 or 22
202		1010	CA	12		0	9	8	2	A830				
203		1011	CB	12		0	9	8	3	A430				
204		1100	CC	12		0	9	8	4	A230				
205		1101	CD	12		0	9	8	5	A130				
206		1110	CE	12		0	9	8	6	A0B0				
207		1111	CF	12		0	9	8	7	A070				
208	1101	0000	D0		11	0				6000	(- zero)			
209*		0001	D1		11				1	5000	J	D1	51 (U)	7C or 7E
210*		0010	D2		11				2	4800	K	D2	52 (U)	58 or 5A
211*		0011	D3		11				3	4400	L	D3	43 (U)	5C or 5E
212*		0100	D4		11				4	4200	M	D4	54 (U)	70 or 72
213*		0101	D5		11				5	4100	N	D5	45 (U)	74 or 76
214*		0110	D6		11				6	4080	O	D6	46 (U)	50 or 52
215*		0111	D7		11				7	4040	P	D7	57 (U)	54 or 56
216*		1000	D8		11			8		4020	Q	D8	58 (U)	64 or 66
217*		1001	D9		11		9			4010	R	D9	49 (U)	60 or 62
218		1010	DA	12	11		9	8	2	C830				
219		1011	DB	12	11		9	8	3	C430				
220		1100	DC	12	11		9	8	4	C230				
221		1101	DD	12	11		9	8	5	C130				
222		1110	DE	12	11		9	8	6	C0B0				
223		1111	DF	12	11		9	8	7	C070				
224	1110	0000	E0			0		8	2	2820				
225		0001	E1		11	0	9		1	7010				
226*		0010	E2			0			2	2800	S	E2	32 (U)	98 or 9A
227*		0011	E3			0			3	2400	T	E3	23 (U)	9C or 9E
228*		0100	E4			0			4	2200	U	E4	34 (U)	B0 or B2
229*		0101	E5			0			5	2100	V	E5	25 (U)	B4 or B6
230*		0110	E6			0			6	2080	W	E6	26 (U)	90 or 92
231*		0111	E7			0			7	2040	X	E7	37 (U)	94 or 96
232*		1000	E8			0		8		2020	Y	E8	38 (U)	A4 or A6
233*		1001	E9			0	9			2010	Z	E9	29 (U)	A0 or A2
234		1010	EA		11	0	9	8	2	6830				
235		1011	EB		11	0	9	8	3	6430				
236		1100	EC		11	0	9	8	4	6230				
237		1101	ED		11	0	9	8	5	6130				
238		1110	EE		11	0	9	8	6	60B0				
239		1111	EF		11	0	9	8	7	6070				
240*	1111	0000	F0			0				2000	0	F0	1A (L)	C4
241*		0001	F1						1	1000	1	F1	01 (L)	FC
242*		0010	F2						2	0800	2	F2	02 (L)	D8
243*		0011	F3						3	0400	3	F3	13 (L)	DC
244*		0100	F4						4	0200	4	F4	04 (L)	F0
245*		0101	F5						5	0100	5	F5	15 (L)	F4
246*		0110	F6						6	0080	6	F6	16 (L)	D0
247*		0111	F7						7	0040	7	F7	07 (L)	D4
248*		1000	F8					8		0020	8	F8	08 (L)	E4
249*		1001	F9				9			0010	9	F9	19 (L)	E0
250		1010	FA	12	11	0	9	8	2	E830				
251		1011	FB	12	11	0	9	8	3	E430				
252		1100	FC	12	11	0	9	8	4	E230				
253		1101	FD	12	11	0	9	8	5	E130				
254		1110	FE	12	11	0	9	8	6	E0B0				
255		1111	FF	12	11	0	9	8	7	E070				

INDEX

C26-5929-2

fold

fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. 452

fold

fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

C26-5929-2

IBM 1130 Printed in U.S.A. C26-5929-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601